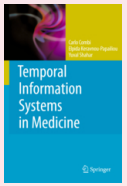
Chapter 4: Temporal Clinical Databases[1]

Carlo Combi[1] Elpida Keravnou-Papailiou[2]
Yuval Shahar[3]

[1]Department of Computer Science, University of Verona

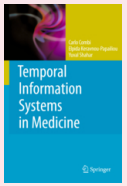
[2]Department of Computer Science, University of Cyprus

[3]Department of Information Systems Engineering, Ben Gurion University

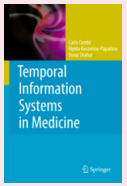


[1]Slides edited with the help of Elena Gaspari, Univ. of Verona

Outline



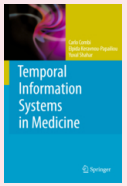
Outline



Introduction

A wide variety of applications and clinical domains 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.

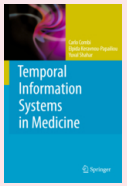
- cardiology;
- oncology;
- psychiatry;
- internal medicine;
- intensive care;
- urology;
- infectious diseases;
- anesthesiology.



TOD

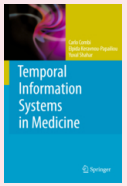
One of the first applications of databases to clinical domains is the **Time Oriented Database (TOD) model**.

- **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)*,
  - *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).



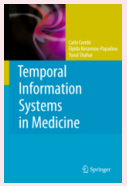
Modeling Temporal Clinical Data

- Often *temporal dimensions* of clinical data are *multiple* and cannot be modeled only through valid and 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



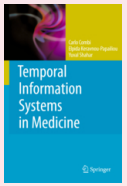
Modeling Temporal Clinical Data (cont.d)

- The temporal dimension may be given at different *granularities* and/or with *indeterminacy*
  - it is possible that in some cases temporal relations cannot be asserted for sure
- Clinical information may consist both of natural language, *qualitative sentences* and of *quantitative parameter values*



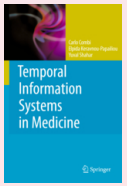
Querying Temporal Clinical Data

- *Several* different *temporal dimensions* could be involved in the definition of a query on temporal clinical data
- The evaluation of clinical information is often performed according to criteria involving *indeterminacy* and *granularities*
- Granularities in queries are not usually related to those used when storing temporal clinical data

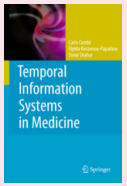


Querying Temporal Clinical Data (cont.d)

- Queries may contain conditions both explicitly related to *absolute time*, and about *qualitative or quantitative temporal relations* between facts
- It is often necessary to consider only clinical data belonging to a specified **temporal window**
  - *absolute*
  - *relative*

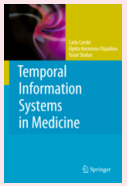


Outline



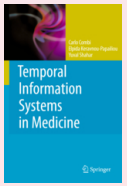
Motivation

- Significant efforts have been undertaken to create database models that can capture the rich semantics of time needed for temporal querying and temporal reasoning in clinical decision support, mainly using the valid time dimension.
- Given the importance of querying time for clinical applications and the lack of standard temporal features in *SQL*, the specification of a standard, rich temporal relational query language for clinical data remains an open challenge.



Main topics in dealing with multiple temporal dimensions of clinical data

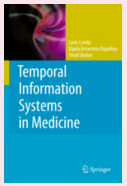
- Introduce the notion of *event time*.
- Introduce a further temporal dimension, called *availability time*.
- Describe a temporal query language, named T4SQL, that extends, and is fully compatible with, SQL and deals with different temporal semantics.



Valid and Transaction Times

Valid time: the *valid time* (VT) of a fact is the time when the fact is true in the modeled reality.

Transaction time: the *transaction time* (TT) of a fact is the time when the fact is current in the database and may be logically retrieved.

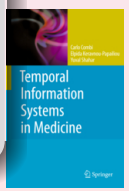


A concrete situation

Example

On August 10, 1998, the physician prescribes a bipuvacaine-based therapy from 10:00 to 14:00. Data about the therapy is entered into the database at 9:00. Due to the unexpected evolution of the patient state, the bipuvacaine infusion is stopped at 11:15 and replaced by a diazepam-based therapy from 11:25 to 14:00. The new facts are entered at 12:00. (Bipuvacaine and diazepam are drugs commonly used in anesthesia.)

Drug	VT	TT
bipuvacaine	[98Aug10;10:00, 98Aug10;14:00)	[98Aug10;9:00, 98Aug10;12:00)
bipuvacaine	[98Aug10;10:00, 98Aug10;11:15)	[98Aug10;12:00, *uc*)
diazepam	[98Aug10;11:25, 98Aug10;14:00)	[98Aug10;12:00, *uc*)

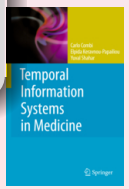


Event Time

Event time: the *event time* (ET) of a fact is the occurrence time of the real-world event that generates the fact.

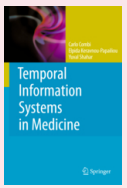
Example

- an on-time promotion event, that increases the rank of a physician, occurs on October 1, 1997, but its effects are recorded in the database on November 1 (delayed update);
- a retroactive promotion event, whose effects are immediately recorded in the database, occurs on November 1, but the increase in rank of the physician becomes valid since October 1 (retroactive update).



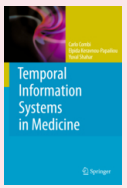
Valid and Event Times

- *On-time events*: the validity interval starts at the occurrence time of the event (e.g., hospitalization starts immediately after the family doctor decision).
- *Retroactive events*: the validity interval starts before the occurrence time of the event (e.g., on December 21, 1997, the manager of the hospital decided an increase of 10% of the salary of the physicians of the Pathology Department, starting from December 1, 1997).
- *Proactive events*: the validity interval starts after the occurrence time of the event (e.g., on January 29, 1998, the family doctor decides the patient hospitalization on February 15, 1998).



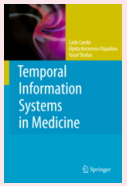
Transaction and Event Times

- *On-time update*: the transaction time coincides with the event time. This situation happens when data values are inserted in the database as soon as they are generated.
- *Delayed update*: the transaction time is greater than the event time. This is the case when data values are inserted some time after their generation.
- *Anticipated update*: the transaction time is less than the event time. This is the case when data values are entered into the database before the occurrence time of the event that generates them. Such a notion of anticipated update is useful to model hypothetical courses of events.



One Event Time is not enough

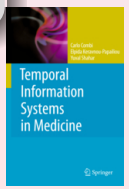
- **IF** there are gaps in temporal validity (a relation modeling admissions to hospital, which only records information about inpatients),
- **OR** only incomplete information about the effects of an event is available (we know that an event that changes the rank of the physician occurred, but we do not know if it is a promotion or a downgrading event),
- **OR** the expected termination of a validity interval is (must be) revised, while its initiation remains unchanged (a prescribed therapy needs to be stopped due to an unexpected evolution of the patient state),
- **THEN** we need to distinguish between the occurrence times of the two events that respectively initiate and terminate the validity interval of the fact.



A concrete situation

Example

On August 10, 1998, at 8:00, the physician prescribes a bipuvacaine-based therapy from 10:00 to 14:00. Data about the therapy is entered into the database at 9:00. Due to the unexpected evolution of the patient state, at 11:00 the physician decides a change in the patient therapy. Accordingly, the bipuvacaine infusion is stopped at 11:15 and replaced by a diazepam-based therapy from 11:25 to 14:00. The new facts are entered at 12:00.

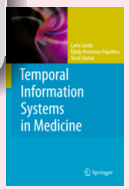


Two (incorrect) solutions

Example

Drug	VT	ET	TT
bipuvacaine	[98Aug10;10:00, 98Aug10;14:00)	98Aug10;8:00	[98Aug10;9:00, 98Aug10;12:00)
bipuvacaine	[98Aug10;10:00, 98Aug10;11:15)	98Aug10;8:00	[98Aug10;12:00, *uc*)
diazepam	[98Aug10;11:25, 98Aug10;14:00)	98Aug10;11:00	[98Aug10;12:00, *uc*)

Drug	VT	ET	TT
bipuvacaine	[98Aug10;10:00, 98Aug10;14:00)	98Aug10;8:00	[98Aug10;9:00, 98Aug10;12:00)
bipuvacaine	[98Aug10;10:00, 98Aug10;11:15)	98Aug10;11:00	[98Aug10;12:00, *uc*)
diazepam	[98Aug10;11:25, 98Aug10;14:00)	98Aug10;11:00	[98Aug10;12:00, *uc*)

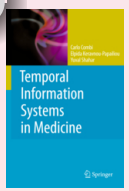


Event time revisited

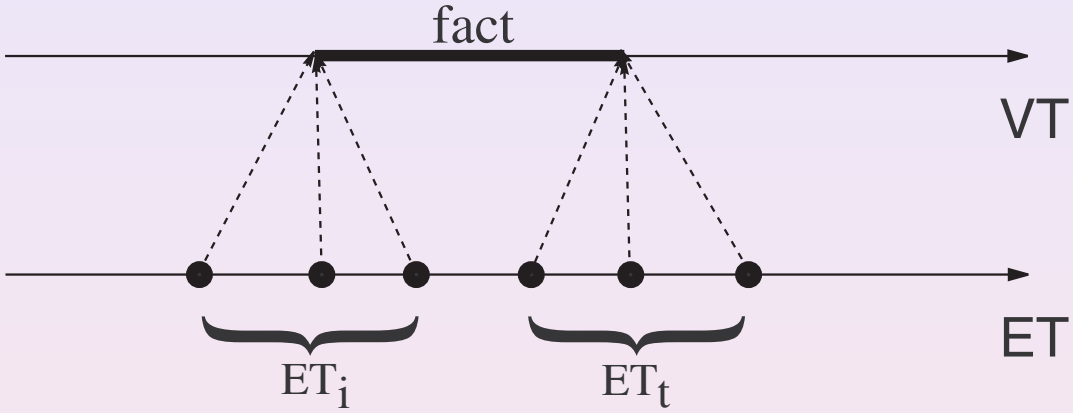
Event time: the *event time* of a fact is the occurrence time of a real-world event that either initiates or terminates the validity interval of the fact.

Example

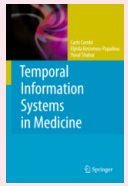
Drug	VT	ET_i	ET_t	TT
bipuvacaine	[98Aug10;10:00, 98Aug10;14:00)	98Aug10;8:00	98Aug10;8:00	[98Aug10;9:00, 98Aug10;12:00)
bipuvacaine	[98Aug10;10:00, 98Aug10;11:15)	98Aug10;8:00	98Aug10;11:00	[98Aug10;12:00, *uc*)
diazepam	[98Aug10;11:25, 98Aug10;14:00)	98Aug10;11:00	98Aug10;11:00	[98Aug10;12:00, *uc*)



Some remarks

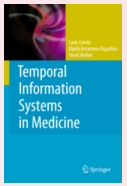


- This distinction between initiating and terminating event times comes from ideas underlying classical formalisms in the area of **reasoning about actions and change**.
- The choice of adding **event time(s) as a separate temporal dimension** has been extensively debated in temporal databases.



Time and Information Systems

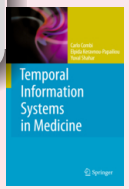
- By **information system** we mean the set of information flows of an organization and the human and computer-based resources that manage them.
- From such a point of view, we may need to model **the time at which** (someone/something within) **the information system becomes aware of a fact** as well as the time at which the fact is stored into the database. While the latter temporal aspect is captured by the transaction time, the former has never been explicitly modeled.



A concrete situation

Example

Due to a trauma that occurred on September 15, 1997, Mary suffered from a severe headache starting from October 1. On October 7, Mary was visited by a physician. On October 9, the physician administered her a suitable drug. The day after, the physician entered acquired information about Mary's medical history into the database. On October 15, the patient told the physician that her headache stopped on October 14; the physician entered this data into the database on the same day. Due to an insertion mistake (or to an imprecision in Mary's talk), the trauma has been registered as happened on September 5, 1997. Only on October 20, the mistake was discovered. The day after, the physician entered the correct data into the database.

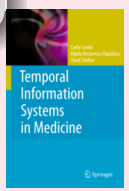


Availability Time

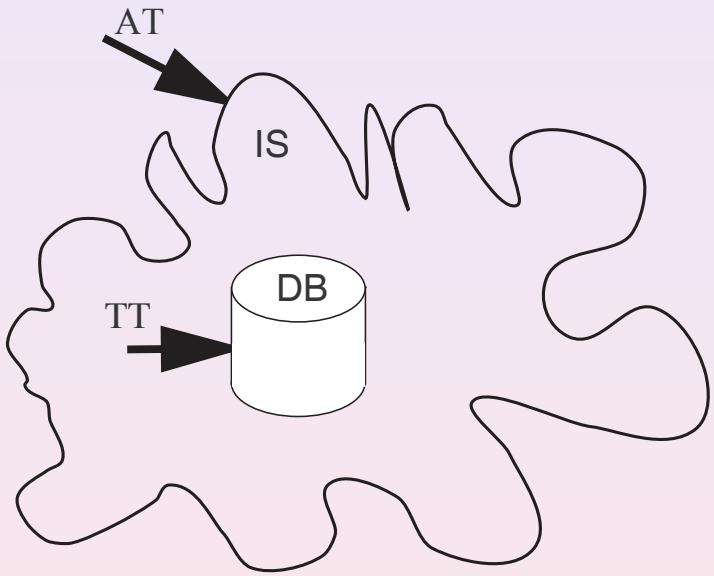
Availability Time: the *availability time* (AT) of a fact is the time interval during which the fact is known and believed correct by the information system.

Example

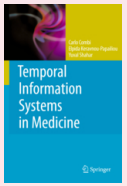
symptom	VT	ET_i	ET_t	AT	TT
headache	[97Oct1, *now*)	97Sept5	null	[97Oct7, 97Oct15)	[97Oct10, 97Oct15)
headache	[97Oct1, 97Oct14)	97Sept5	97Oct9	[97Oct15, 97Oct20)	[97Oct15, 97Oct21)
headache	[97Oct1, 97Oct14)	97Sept15	97Oct9	[97Oct20, *uc*)	[97Oct21, *uc*)



Transaction and Availability Times

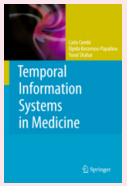


- The availability time can be viewed as **the transaction time of the information system.**



Transaction and Availability Times

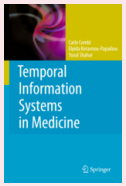
- If the information system outside the database is considered a database (in general, this is not the case; in our medical examples, for instance, the information system includes both databases and physicians), then the availability time of a fact is when the fact is inserted into (deleted from) the information system.
- This analogy between availability time and transaction time makes it immediately clear why the availability time has to be an interval.



The complete picture

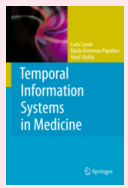
Modeled reality	***Information system***
valid time	transaction time
↑	↑
event times	availability time

- Transaction time is append-only.
- Valid and event times, being related to times of the represented real world, can be located either in the past or in the future, and they can be modified freely.



The complete picture (cont'd)

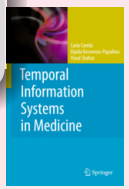
- The availability time is append-only in its nature, because facts previously known and believed correct by the information system cannot be changed.
- However, from the point of view of the database system, availability time would be really append-only only if there were no errors in data entry.
- Since we cannot exclude such a possibility, previous states of the information system, according to the availability time, can be revised by entering new data.
- Furthermore, even assuming data entry without errors, database and transaction times may "append" facts according to two different orders.



A clinical example

Example

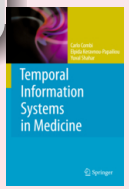
- Patient Hubbard, identified by the attribute *PatId* having value *p*2, was visited on June 30, 2008 and high systolic and diastolic blood pressures were revealed (an SBP of 170 mm Hg and a DBP of 120 mm Hg were measured several times that day), due to a strong emotional stress when he was driving 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.



A clinical example (cont.d)

Example

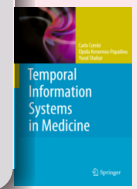
- On June 30, the physician prescribed atenolol (50 mg/day) to the patient since that moment. Due to insertion mistakes, data represent a wrong dosage (5 mg/day) and a wrong start time of the therapy (June 30, 2007). These (wrong) data were inserted on July 2 into the database.
- On July 4, patient Hubbard was visited again and his blood pressure had normal values. The physician stored immediately this information into the database.



A clinical example (cont.d)

Example

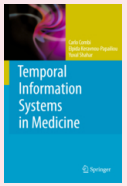
- On July 12, patient Hubbard calls the physician and says him that he stopped the therapy on July 11, due to an increasing chest pain since the day before: the physician prescribed to the patient a therapy with lisinopril, 10 mg/day until July 15, when the patient will need a further therapy redefinition.
- 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.



A clinical example (cont.d)

The example highlights several and different temporal dimensions.

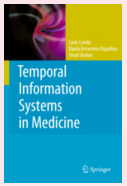
- AT: the moment when the physician becomes aware of the patient's situation (e.g., hypertension),
- TT: the moment the corresponding data are inserted into the database,
- ET_i: the moment when the physician decides for a therapy,
- ET_t: the moment when the patient decides for the therapy interruption.



A clinical example (cont.d)

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.



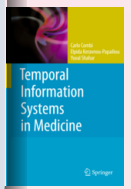
The Temporal Data Model

The temporal data model considers all the four temporal dimensions and is a straightforward extension of the relational model and encompasses a single type of key constraints, namely, the *snapshot key* constraint.

Definition

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:

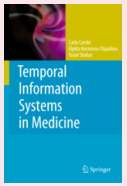
- $\forall a \in K(t[a] \neq \texttt{null})$
- $\forall t1, t2((t1[VT, TT, AT] \cap t2[VT, TT, AT] \neq \emptyset \wedge t1 \neq t2) \Rightarrow t1[K] \neq t2[K])$



The Temporal Data Model

The model imposes some basic constraints on the relationships between the values of the various temporal dimensions:

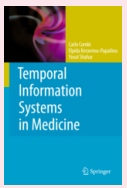
- we cannot assign a value to ET if there exists no value for the corresponding VT;
- we cannot assign a value to AT if there exists no value for the corresponding TT.



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]	....

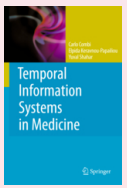
	ET_i	ET_t	AT	TT
....	2008Jun30		[2008Jun30, 2008Jul12]	[2008Jul02, 2008Jul12]
....	2008July12	2008Jul12	[2008Jul12, *uc*]	[2008Jul13, *uc*]
....	2008Jun30	2008Jul10	[2008Jul12, *uc*]	[2008Jul13, 2008Jul16]
....	2008Jun30	2008Jul10	[2008Jul12, *uc*]	[2008Jul17, *uc*]



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

PatId	SBP	DBP	VT	
*p*1	120	80	[2008May01, 2008May01]	..
*p*2	170	120	[2008Jun30, 2008Jun30]	....
*p*2	115	70	[2008July04, 2008Jul04]	....

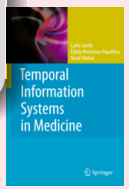
	ET_i	ET_t	AT	TT
....	2008May01	2008May01	[2008Nov17, *uc*]	[2008Nov17 *uc*]
....	2008Jun28	2008Jun30	[2008Jun30, *uc*]	[2008Jul02, *uc*]
....	2008Jun30	2008Jul04	[2008July04, *uc*]	[2008Jul04, *uc*]



The Clinical Database

Example

- Relation `PatTherapy` stores information about patients (attribute `PatId`) and prescribed therapies (attributes `Drug`, `DailyDose`, and `Unit`).
- 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.

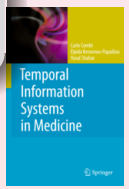


The Clinical Database (cont.d)

Example

Let us consider some relevant information we are allowed to derive from these data in a decision making/decision evaluation perspective:

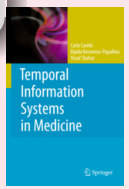
- According to the current content of the database, patient *p*2 assumed atenolol from June 30, 2008 to July 11, 2008.
- The event initiating this therapy happened on June 30.
- The event finishing the therapy happened on July 10, one day before the end of the therapy.



The Clinical Database (cont.d)

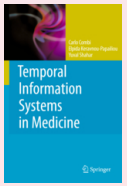
Example

- 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.



T4SQL semantics

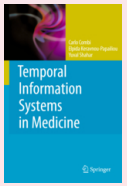
- current, which considers only current tuples;
- sequenced, which corresponds to the homonymous SQL3 semantics;
- atemporal, which is equivalent to the SQL3 non-sequenced one;
- next, which allows one to link consecutive states when evaluating a query.



T4SQL semantics (cont.d)

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 consider complete (i.e., with all the four temporal dimensions) tables in the FROM clause: queries return relations with at most the four temporal dimensions.
- Tables without all the four temporal dimensions are prior converted to complete relations according to either some default rules or other alternative rules, suitably defined according to the given application domain.

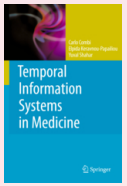


T4SQL Time Data Types

- The only data type used to describe an instant is DATE.
- the PERIOD data type is defined by instants of type DATE.
- Interval data types (i.e., representing durations) are day and year-month.

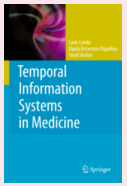
Values associated to a specific temporal dimension can be referenced by the following functions, only:

1. **VALID(T)** returns the *VT* of a tuple of table R;
2. **TRANSACTION(T)** returns the *TT* of a tuple of R;
3. **AVAILABLE(T)** returns the *AT* of a tuple of R;
4. **INITIATING_ET(T)** returns ET_i for a tuple of R;
5. **TERMINATING_ET(T)** returns ET_t for a tuple of R.



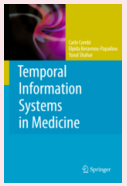
T4SQL syntax

```
[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>]
```



T4SQL syntax (cont.d)

```
<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
```



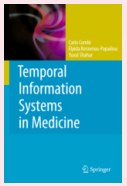
The Clause SEMANTICS

T4SQL enables the user to specify different semantics for every temporal dimension:

```
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.
- Tokens ON and TIMESLICE are optional and aim at increasing the readability of the query.
- `<ts_exp>` is a constant (p) either of type `PERIOD` or of type `DATE`



The Clause SEMANTICS (cont.d)

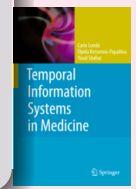
Example

Consider table `PatTherapy` and the need of extracting:

- data about all the patients that had a therapy with atenolol;
- the respective period during which the fact happened.

When temporal dimensions are managed explicitly by the user, the query is the following:

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



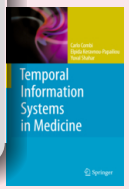
The Clause SEMANTICS (cont.d)

- 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.

Example

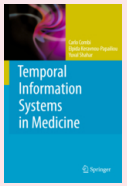
The previous query is expressed in T4SQL as follows:

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



ATEMPORAL Semantics

- With `ATEMPORAL` semantics, the corresponding attribute(s) is dealt with as atemporal (timeless), providing the user with the highest level of freedom in managing temporal dimensions.

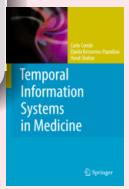


ATEMPORAL Semantics

Example

We want to retrieve all the patients where the therapy "`Atenolol`" has never been observed.

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



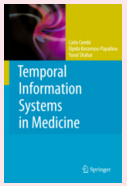
CURRENT Semantics

Definition

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:

- 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'`.

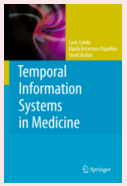


CURRENT Semantics

Example

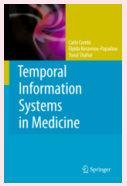
We want to retrieve all the patients from `PatVisit`, as currently stored. The T4SQL query is:

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



SEQUENCED Semantics

- The `SEQUENCED` semantics forces a *time point by time point* evaluation of the statement.
- This evaluation on a given temporal dimension *d* considers all the tuples of tables in the `FROM` clause where there exists a date *i* belonging to all the periods of the dimension *d*.
- As a difference from the `ATEMPORAL` and `CURRENT` semantics, the `SEQUENCED` semantics returns in the result table the temporal dimension specified in the query.

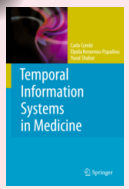


SEQUENCED Semantics (cont.d)

Example

We want to retrieve for every patient the visits whose VT overlaps the VT of the prescribed therapies.

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

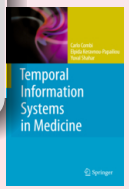


SEQUENCED Semantics (cont.d)

Example

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
```

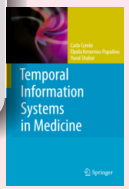


SEQUENCED Semantics (cont.d)

Example

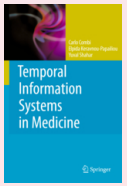
In this case, too, the query can be translated to a different one with the `ATEMPORAL` semantics, as follows:

```
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'
```



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.
- 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

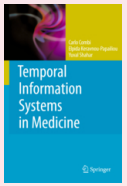


NEXT Semantics (cont.d)

The complete syntax for the `NEXT` semantics is defined as:

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

- `<duration>` is the width of the temporal interval where the successor must be found
- `<dimension>` is the dimension where the semantics must be applied to.

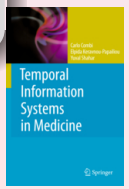


NEXT Semantics (cont.d)

Example

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
```

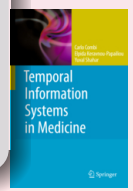


NEXT Semantics (cont.d)

Example

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:

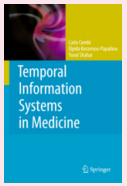
```
SEMANTICS NEXT(0) ON VALID
SELECT    PatId
FROM      PatTherapy AS t
WHERE     t. Drug = 'Atenolol' AND
          t.DailyDose = 50 AND t.Unit = 'mg' AND
          NEXT(t).DailyDose = 70 AND
          NEXT(t).Unit = 'mg'
```



The Clause SELECT

T4SQL introduces the token `WITH` in the `SELECT` clause, to separate the specification of explicit attributes from that of implicit attributes.

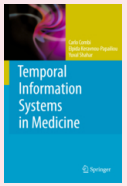
- Any statement before the token `WITH` is evaluated according to the SQL92 semantics for projection;
- any statement following the token `WITH` computes the temporal dimension(s) to be included in the result relation.



The Clause SELECT: syntax

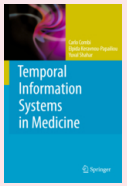
```
SELECT  <sel_element_list>
 [WITH <w_exp> [AS] <dim> {, <w_exp> [AS] <dim>}]
```

- `<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)
- `<dim>` is a temporal dimension
- the optional token `AS` increases the readability of the code



The Clause SELECT: Default Temporal Dimensions

- `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.



The Clause SELECT: Default Temporal Dimensions (cont.d)

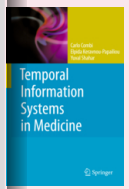
Example

We want to retrieve patients who

- had high diastolic blood pressure (i.e., more than 100 mmHg),
- received the drug *'Lisinopril'*,
- 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 a specific VT: from the beginning of the high diastolic blood pressure to the end of the therapy[a].

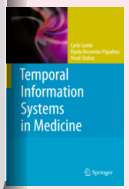
[a] the corresponding query on the book needs to be revised



The Clause SELECT: Default Temporal Dimensions (cont.d)

Example

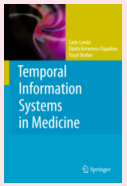
```
SEMANTICS ATEMPORAL ON VALID
SELECT   PatId WITH PERIOD (BEGIN(VALID(pv1)),
           END(VALID(pt))) AS VALID
FROM     PatVisit AS pv1, PatVisit AS pv2,
         PatTherapy AS pt
WHERE    pv1.DBP > 100 AND pv2.DBP < 100 AND
         pt. Drug = 'Lisinopril' AND
         pv1.PatId = pt.PatId AND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



The Clause SELECT: Coalescing

- In a temporal relation, all the tuples have to satisfy the snapshot key constraint.
- 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*

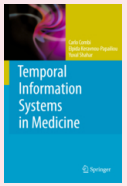
To cope with this situation, the **coalescing** operator fuses the tuples with overlapping values of temporal dimensions and with the atemporal attributes having the same corresponding values.



The Clause FROM

The `FROM` clause of SQL92 specifies tables involved in the query, and may contain the specification of join operations.

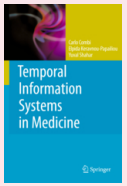
- T4SQL adopts the same `FROM` clause as SQL92.
- Additionally, the token `JOIN` of SQL92 has been replaced by the token `TJOIN` (*temporal join*), when some join conditions are temporal.



The Clause FROM (cont.d)

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.

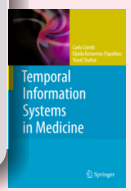


The Clause FROM (cont.d)

Example

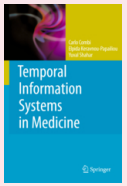
We want to select, according to the current state of the database, all the patients, reporting also their vital signs, who had a visit during the assumption of a therapy that ended more than 10 days after the visit.

```
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 )
```



The Clauses WHERE and WHEN

- 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.

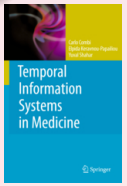


The Clause FROM (cont.d)

Example

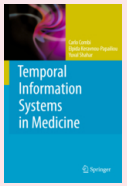
Consider again the query of the previous example.

```
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
```



Clauses GROUP BY and HAVING

- In a temporal query language, the `GROUP BY` clause can be used to implement a *temporal grouping*.
- The selection of the tuples to be grouped together is performed according to
  - the value of the considered temporal attribute(s) of the tuple
  - the periods associated with the partition
  - the temporal comparison operator used in the temporal grouping



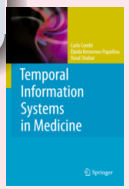
Temporal Grouping

Definition

Let P be a period associated with a particular element of the partition

- P_t be the value assumed by the tuple t on the considered temporal attribute,
- 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.

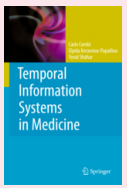


Temporal Grouping (cont.d)

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 is $P \cap P_t$.



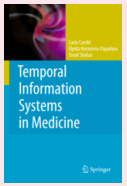
Temporal Grouping (cont.d)

The token `USING` distinguishes between a classic (atemporal) grouping and a temporal grouping.

`GROUP BY <temp> USING MONTH`

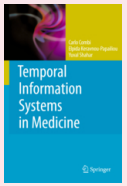
Implemented temporal partitions are:

- `SECOND`,
- `MINUTE`,
- `HOUR`,
- `DAY`,
- `MONTH`,
- `YEAR`.



Temporal Grouping (cont.d)

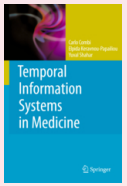
- For the functions `MAX`, `MIN`, `AVG` and `SUM` the token `WEIGHTED` is introduced by T4SQL.
- The token `WEIGHTED` must precede the aggregation function it is applied to.



Temporal Grouping (cont.d)

Let us assume that:

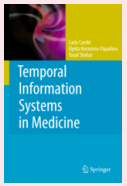
- $t_1 \ldots t_n$ are the tuples belonging to the group G,
- $t_1(d) \ldots t_n(d)$ are the values of the tuple over the temporal attribute d according to which we performed the temporal grouping,
- $d(G)$ is the duration of the considered grouping partition,
- moreover, let $t_1(a) \ldots t_n(a)$ be the values of the tuples for attribute a which is the parameter for the weighted aggregate function.



Temporal Grouping (cont.d)

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)` $= \frac{\sum_{i=1}^{n}(t_i(a) * \frac{duration(t_i(d))}{d(G)})}{n}.$

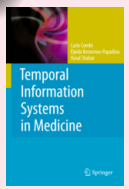


Temporal Grouping (cont.d)

Example

Let us assume that we want to retrieve for every year the average duration of prescribed therapies.

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



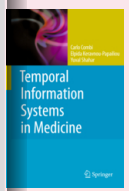
Temporal Grouping (cont.d)

Example

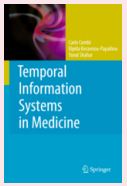
We want to compute from the table `PatVisit` the average level per month of DBP for each patient, returning only data for patients having assumed a (weighted) average quantity of atenolol per month more than 2 mg per day[a].

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

[a] the corresponding query on the book needs to be revised

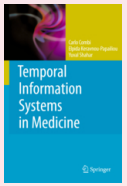


Outline



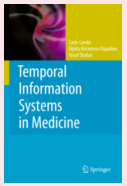
Granularity and Indeterminacy in Clinical Databases

- Since the beginning of 1990s, several work in the medical informatics field focused on the issue of **temporal granularity and indeterminacy in modeling and querying clinical data**.
- In the following we will describe two well known methodologies adopting the *relational model* and the *object-oriented one*, respectively.



A Temporally Extended Clinical Relational System

- One of the main problems: 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.



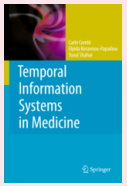
A Temporally Extended Clinical Relational System

Limitation in the clinical domain:

In a relation containing descriptions of pathologies:

- instantaneous tuples (e.g., “cerebral stroke at 21:23 of May 16th, 1997”)
- interval-based tuples (e.g., “vision loss on February 13th 1997 from 18:35:15 to 18:45:28”)

must co-exist.

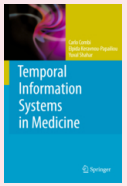


A Temporally Extended Clinical Relational System: Chronus

Chronus distinguishes two kinds of medical temporal data:

- instantaneous data represent *events*
- interval-based data represent *states*

For managing the *indeterminacy* of events and states, Das and Musen define four different types of relational tuples: **event**, **start**, **body**, and **stop** tuples.



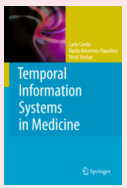
The Chronus System

Relation *Patient*

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

Relation *Visit*

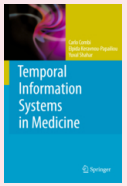
Start_time	Stop_time	Type	PatId	SBP	DBP
98/6/6/11/00	98/6/6/11/59	event	SM1	130	90
98/11/10/10/00	98/11/10/10/09	event	SM1	120	70
99/7/20/12/45	99/7/20/12/59	event	RS1	150	110
97/6/12/00/00	97/6/12/11/59	event	HB3	80	60



The Chronus System

Relation *Therapy*

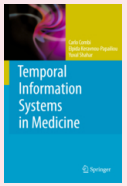
Start_time	Stop_time	Type	PatId	drug	dosage
98/5/12/00/00	98/5/22/23/59	body	SM1	thiazide diuretics	30 mg once a day
99/7/20/00/00	99/7/20/23/59	start event	RS1	aspirin	120 mg daily
99/7/21/00/00	99/7/31/23/59	body	RS1	aspirin	120 mg daily
99/8/1/00/00	99/8/31/23/59	stop event	RS1	aspirin	120 mg daily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



A Temporally Extended Clinical Relational System: Chronus

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

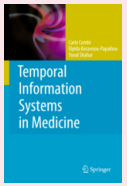
- compatibility with the flat relational model and with SQL;
- addition and enhancement of some specific clauses, functions and predicates of temporal query languages.



A Temporally Extended Clinical Relational System: Chronus

With respect to these needs and to the requirements related to the clinical domain, there are two main approaches:

- proposal of an extended SQL syntax, which is compatible also for querying standard SQL tables;
- design and implementation of software modules for providing physicians with a more simple temporal query language, which is in turn based on SQL.

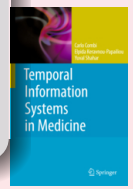


The Chronus **Time Line SQL (TLSQL)**

Example

Considering tables *Patient*, *Visit*, and *Therapy*, we want to retrieve all the patients' visits that occurred during a therapy lasting for sure more than 7 days.

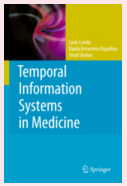
```
SELECT Name, SBP
FROM   Visit V, Therapy T, Patient P
WHEN   [V.Start_time, V.Stop_time]  DURING
                [T.Start_time, T.Stop_time] AND
       DURATION([T.Start_time, T.Stop_time])
            > 7 DAYS AND
       V.Type = "event" AND T.Type = "body"
WHERE  V.PatId = T.PatId AND P.PatId =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



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.
- Medical records are complex documents, composed by different kinds of multimedia data, often involving strong temporal aspects.

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.

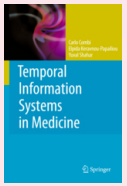


The Temporal Data Model GCH-OODM

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

GCH-OODM Types:

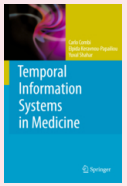
- `string`
- `int`
- `real`
- `set<t>`
- `bag<t>`
- `list<t>`
- `array<t>`



The Temporal Data Model GCH-OODM

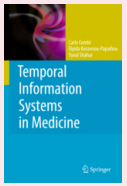
GCH-OODM predefined data types for time and temporal dimensions:

- the type hierarchy `el_time`, `instant`, `duration`, `interval`;
- the collection type `t_o_set`, and some of its specializations, by which sets of temporal objects are modeled.



The Temporal Data Model GCH-OODM

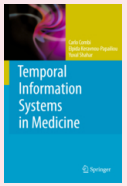
- The basic time domain **T**, called also *time axis*, is isomorphic to the natural numbers with the usual ordering relation $\leq$.
- The set *Gran* of granularity mappings is related to granularities of the Gregorian calendar (years, months, days, hours, seconds).
- Granularity mappings consider granularities for both anchored and unanchored time spans.



The Temporal Data Model GCH-OODM

The type `el_time` allows one to model time points on the basic time axis, named elementary instants

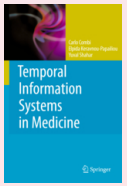
- Each elementary instant is the basic unit of time supported by the temporal DBMS.
- Both time points and spans between time points are modeled in a homogeneous way: time points are identified on the basic time axis by their distance from the origin of the axis.



The Temporal Data Model GCH-OODM

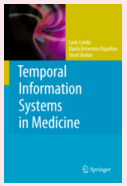
Two different formats for anchored time and unanchored time spans.

- The calendric notation `YY/MM/DD/HH:Mi:SS` allows the specification of a time point.
- 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).



GCH-OODM: Instants, durations, and intervals

- An *interval* is represented by its starting instant, duration and ending instant.
- *Instants* and *duration*, expressed at different granularities/indeterminacy, are based on a discrete time axis, where points are represented by the finest time unit considered by the model.

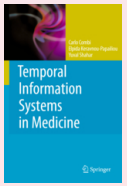


GCH-OODM: Instants, durations, and intervals (cont.d)

Observation

The type `instant` represents 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.

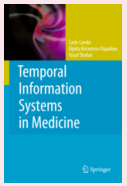


GCH-OODM: Instants, durations, and intervals (cont.d)

Observation

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.

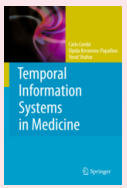


GCH-OODM: Instants, durations, and intervals (cont.d)

Observation

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.



GCH-OODM: Notations for intervals

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

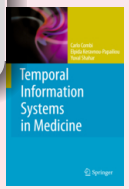
Example

⟨1994/10⟩

- **notation 2.** ⟨x.start(), x.end()⟩.

Example

⟨98/6/6, 99/3/12/13⟩



GCH-OODM: Notations for intervals (cont.d)

- **notation 3.** ⟨x.start(), x.dur()⟩

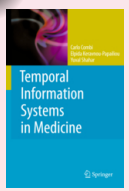
Example

⟨98/6/6, 3 h⟩

- **notation 4.** ⟨x.dur(), x.end()⟩

Example

⟨33 h, 96/10⟩



GCH-OODM: Notations for intervals (cont.d)

- **notation 5.** ⟨in, x.dur()⟩, where in is a granule

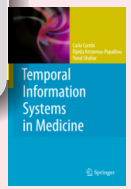
Example

⟨⟨98/6⟩, 7 mi 33 s ⟩

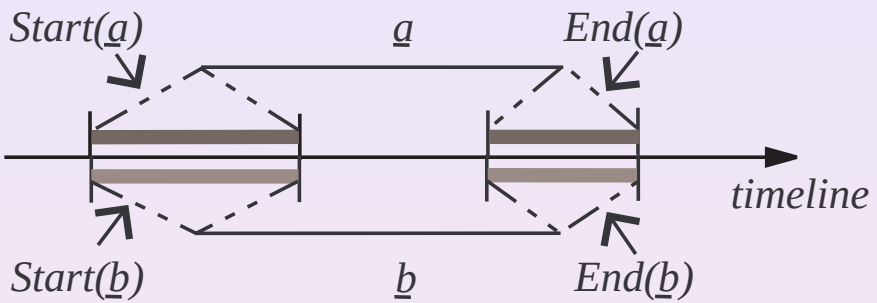
- **notation 6.** ⟨x.start(), x.dur(), x.end()⟩

Example

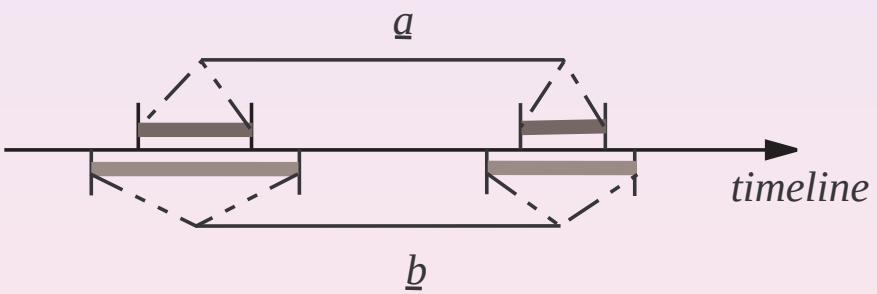
- ⟨97/8/9, ≺5 h 5 mi 2 s, 24 h 8 mi 3 s≻, 97/8/10⟩
- ⟨≺96/4/3/12/30/10, 96/4/3/23/30/0≻, ≺4 h 6 mi 3 s, 24 h 35 mi 2 s≻, 96/4/4⟩



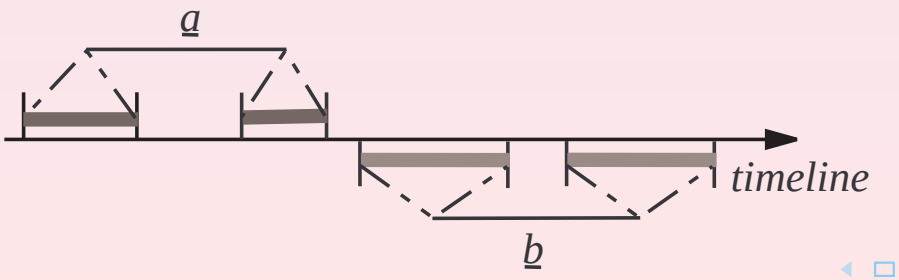
Intervals: Non-Granularity Based Relationships (1)



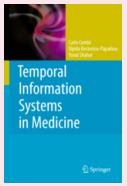
$a \approx b$



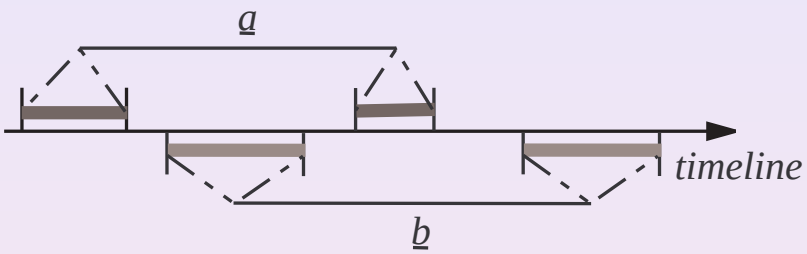
a **tS** b



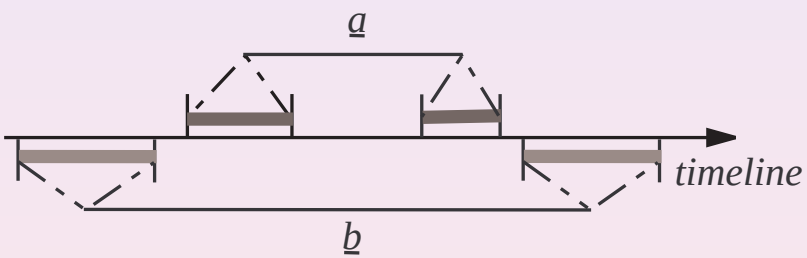
$a < b$



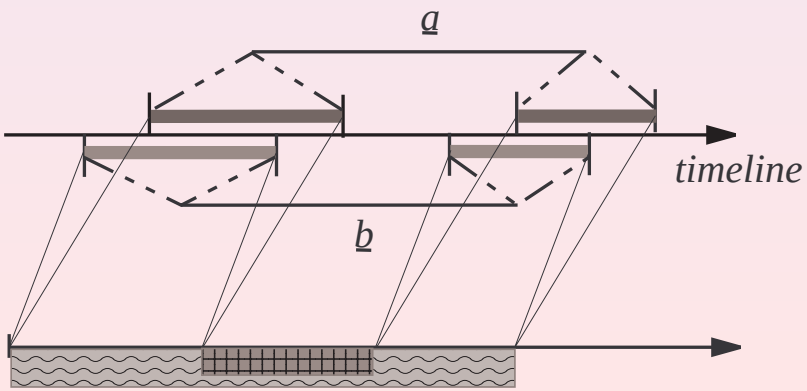
Intervals: Non-Granularity Based Relationships (2)



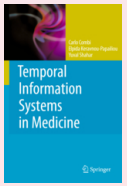
$\underline{a}$ **O** $\underline{b}$



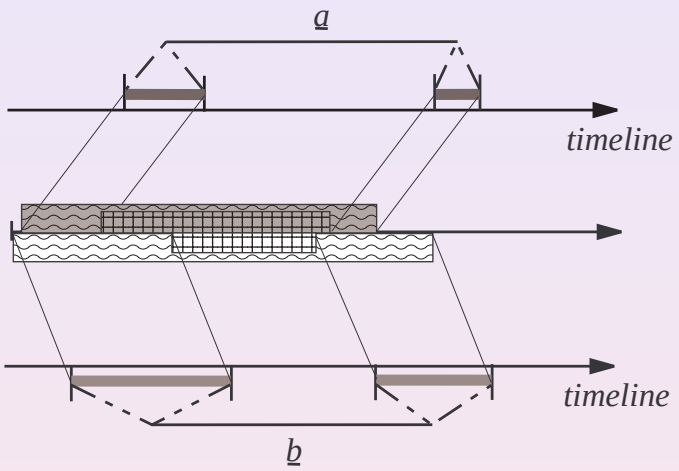
$\underline{a}$ **D** $\underline{b}$



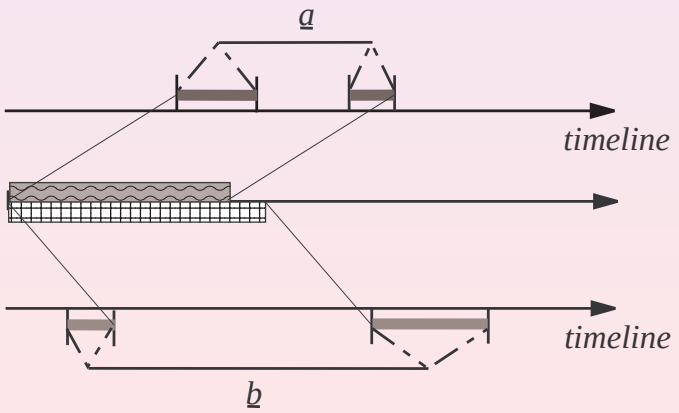
$\underline{a}$ **Dur** $\approx$ $\underline{b}$



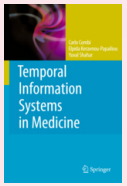
Intervals: Non-Granularity Based Relationships (3)



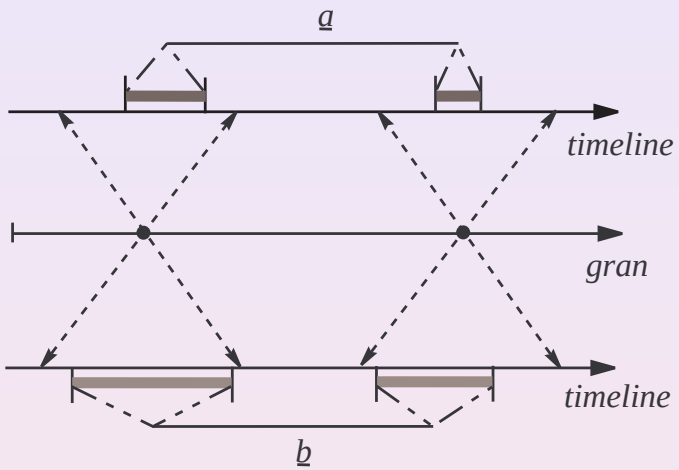
$\underline{a}$ **DurS** $\underline{b}$



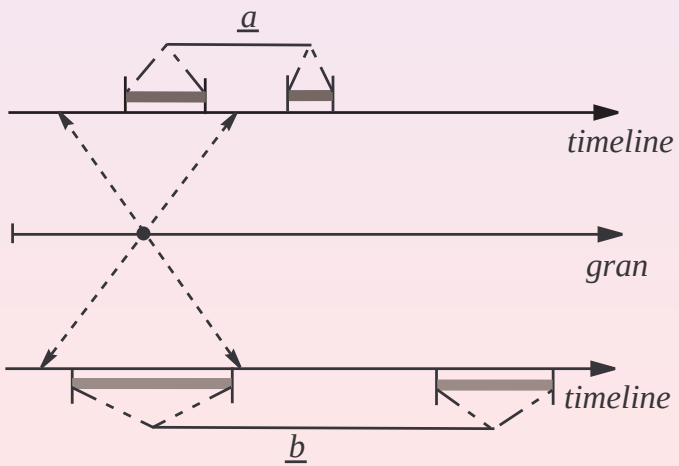
$\underline{a}$ **Dur** $<$ $\underline{b}$



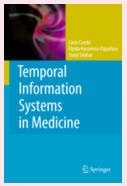
Intervals: Granularity Based Relationships (1)



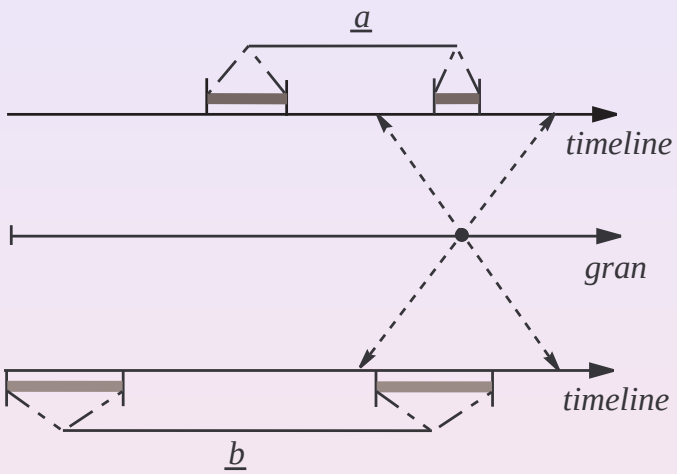
$$\underline{a} =_{gran} \underline{b}$$



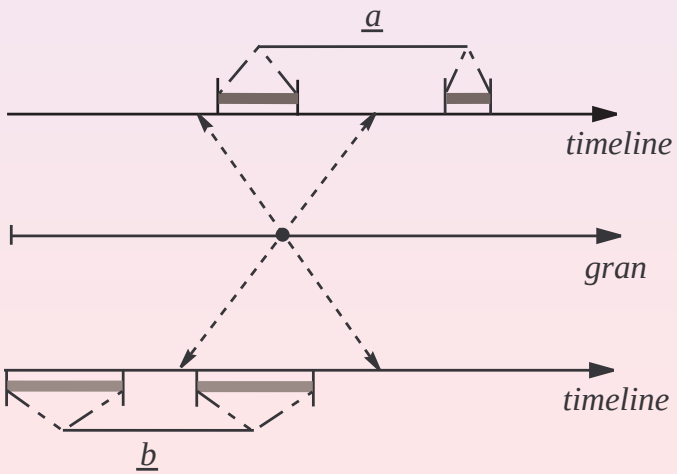
$$\underline{a}\ \mathbf{S}_{gran}\underline{b}$$



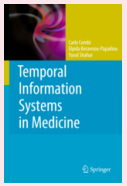
Intervals: Granularity Based Relationships (2)



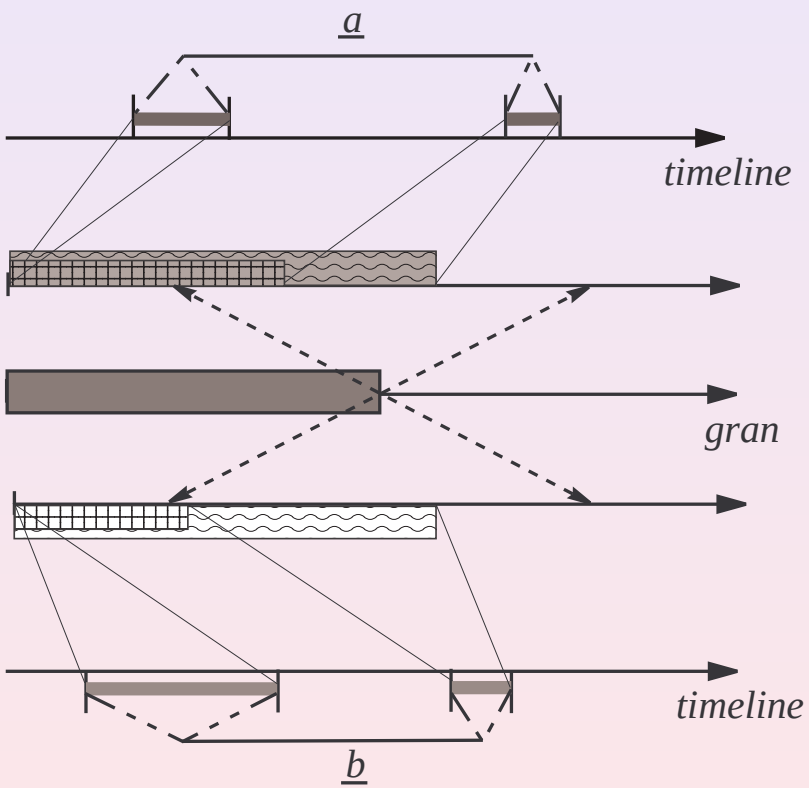
$\underline{a}\ \mathbf{F}_{gran}\underline{b}$



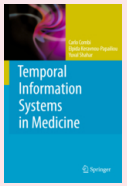
$\underline{a}\ \mathbf{M}_{gran}\underline{b}$



Intervals: Granularity Based Relationships (3)



$\underline{a}$ **Dur**$_{gran}$ $\underline{b}$



The Temporal Data Model GCH-OODM

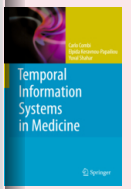
The presence of different granularities/indeterminacies leads to manage relations having logical values *True*, *False*, and *Undefined.*

Example

Let us consider the two sentences:

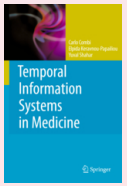
- "In July 1998 the patient suffered from headache for eight days",
- "In July 1998 the patient had fever for 17 days".

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.



The Temporal Data Model GCH-OODM

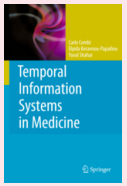
- GCH-OODM uses a *three-valued logic* with values `T`: *True*, `F`: *False*, and `U`: *Undefined*.
- The usual logical connectives AND, OR, NOT, IMPLIES, ..., and the logical quantifiers *EXISTS* ($\exists$), and *FOR EACH* ($\forall$) have been extended to consider the third truth value *Undefined*.
- The new logical connectives `T()`, `U()` and `F()` explicitly manage each of the three truth values.



GCH-OODM Truth Tables for Connectives

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

A	(NOTA)	T(A)
T	F	T
F	T	F
U	U	F



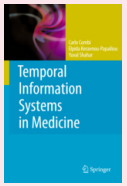
GCH-OODM Formulas

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;

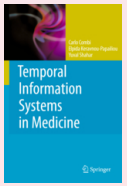
c) composition by the logical connectives of formulas of type a) or b).



GCH-OODM: 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.
  - Method *validInterval()* returns the interval of validity of an object.
- Objects instances of atemporal types (hereinafter atemporal objects) model information, not having an associated temporal dimension.
  - An atemporal object can, however, have properties represented by temporal objects.

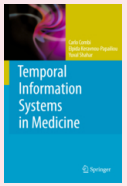


GCH-OODM: Temporal and atemporal types (cont.d)

Both in temporal and atemporal types we distinguish, then,

a) **temporal methods**, modeling temporal features, returning temporal objects,

b) **atemporal methods** returning atemporal objects.



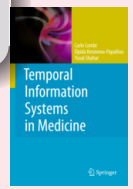
GCH-OODM: Temporal and atemporal types (cont.d)

- In GCH-OODM temporal properties are modeled by temporal objects, which can be composed by a set of temporal objects.
- It is possible to model constraints on the valid time of objects and properties:

Example

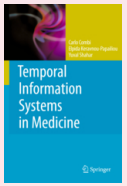
Let `o.p()` be an object modeling a temporal property of a temporal object `o`; the following relation must hold:

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



GCH-OODM: Sets of Temporal Objects

- 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.



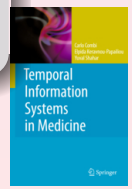
GCH-OODM: Sets of Temporal Objects

Example

Let us consider 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*.



GCH-OODM: Sets of Temporal Objects (cont.d)

Example

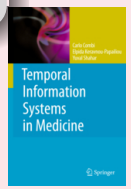
- *OCCURS(p)*

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

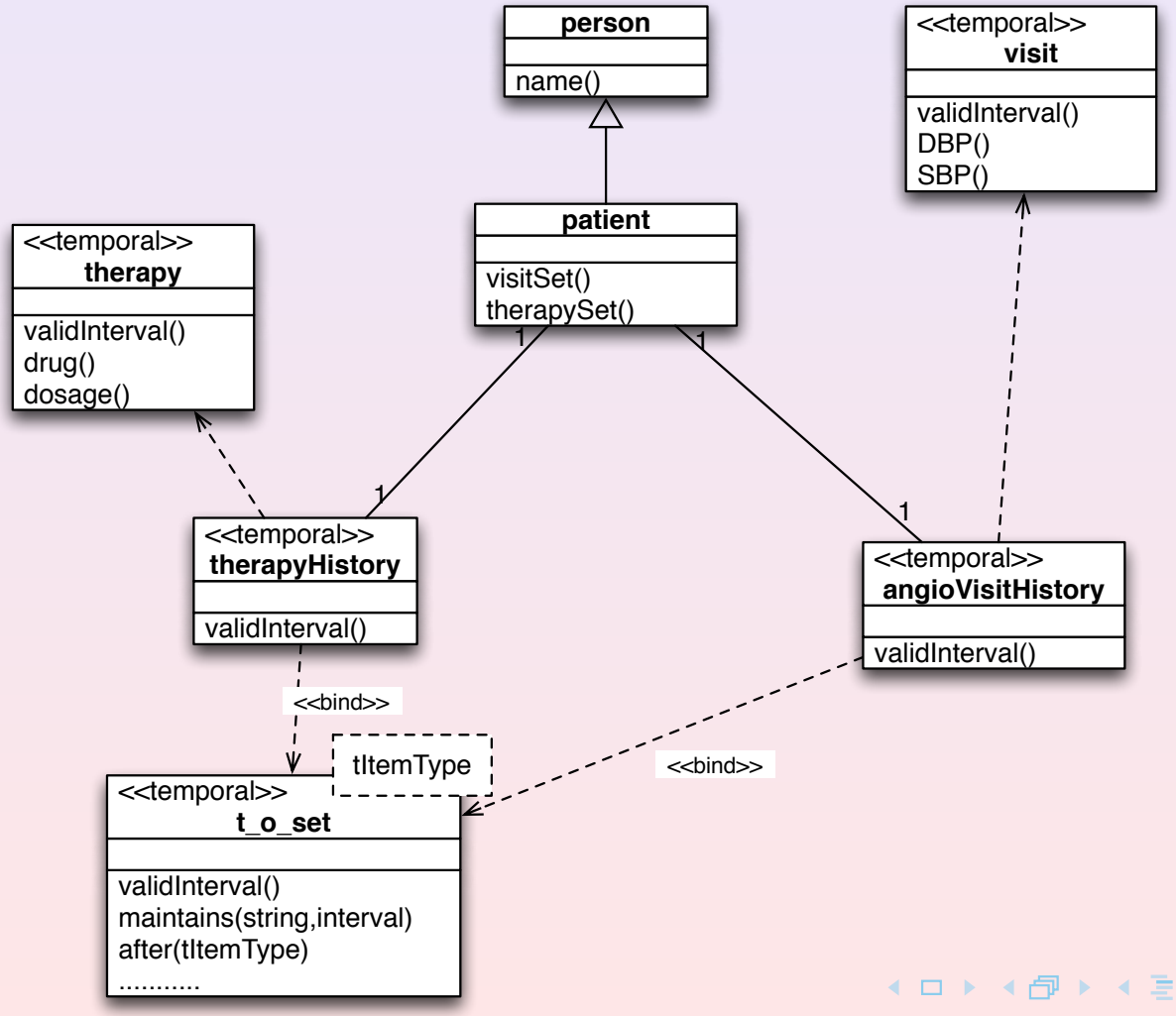
- *CONTEMPORARY(p, q, X)*

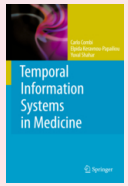
$$I.CONTEMPORARY(p, q, X) \equiv \exists x \in I.subset(p),$$

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



GCH-OODM: the example database





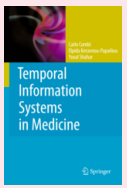
GCH-OODM: a database instance

OID	name()
p1	Smith
p2	Hubbard
p3	Rossi

a) objects of type `person`

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

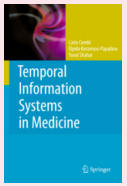
b) objects of type `patient`



GCH-OODM:a database instance (cont.d)

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`



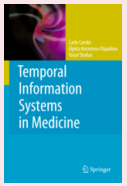
GCH-OODM:a database instance (cont.d)

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`

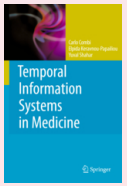
OID	validInterval()	drug()	dosage()
t1	⟨98/5/11, 98/5/23⟩	thiazide diuretics	30 mg once a day
t2	⟨99/7/20, 99/8⟩	aspirin	120 mg daily
t3	⟨99/7/21/16, 5 dd⟩	heparin	18 units/kg/hr

e) objects of type `therapy`



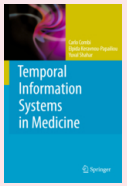
The Temporal Query Language GCH-OSQL

- 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.



GCH-OSQL syntax

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

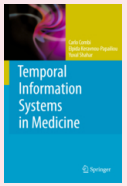


GCH-OSQL semantics

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 conditions imposed through the optional clauses `WHERE`, `TIME-SLICE`, `MOVING WINDOW`.

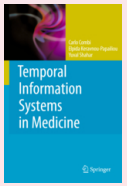
Retrieved objects are those for which the specified conditions result in `TRUE` or `UNDEFINED` logical values.



The SELECT FROM WHERE clauses

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

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

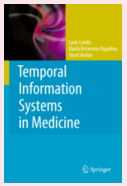


The SELECT FROM WHERE clauses (cont.d)

Example

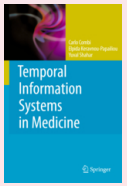
Find all the vital signs measured during visits; display all data about visits and also the name of visited patients.

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



The SELECT FROM WHERE clauses (cont.d)

- In the WHERE clause logical conditions express the constraints that must be satisfied by the selected objects
- Conditions involving temporal relations are expressed in the WHERE clause through methods of types `instant`, `duration`, `interval`, and `t_o_set`.

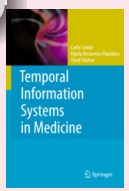


The SELECT FROM WHERE clauses (cont.d)

Example

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.

```
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
```

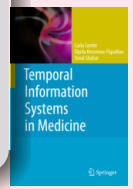


The SELECT FROM WHERE clauses (cont.d)

Example

Find the patients having had a therapy with thiazide diuretics and with aspirin, while aspirin-based therapy surely occurred before that with diurectis.

```
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())
```

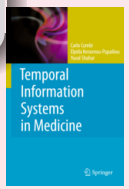


The SELECT FROM WHERE clauses (cont.d)

Example

Find the patients having had a therapy with thiazide diuretics and with aspirin, while aspirin-based therapy surely occurred before that with diurectis.

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


  ```

The TIME-SLICE clause

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

- by the `FROM..TO` keywords

Example

```
FROM 1994/12/11 TO 1994/12/23/11:00.
```

- by the `FROM..FOR` keywords

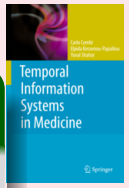
Example

```
FROM 1994/12/11 FOR 2 mm.
```

- by the `FOR..TO` keywords

Example

```
FOR 3 dd TO 1995/4.
```

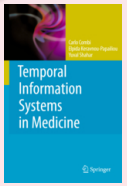


The TIME-SLICE clause (cont.d)

Example

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.

```
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≻
```

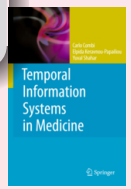


The MOVING WINDOW clause

Example

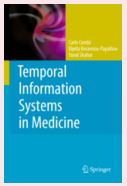
Show the name of patients having had diastolic blood pressure greater than 120 and therapies with diuretics in a period of fifteen days.

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



The MOVING WINDOW clause (cont.d)

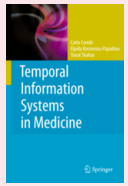
In the `MOVING WINDOW` clause through the `MUST` or `MAY` keywords, only, respectively, certain or uncertain situations can be considered.



Query Processing: Main Steps I

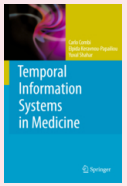
Different logical steps can be identified in the evaluation of a GCH-OSQL query.

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.
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.



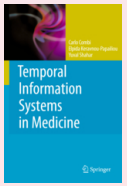
Query Processing: Main Steps II

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.
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).



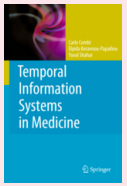
Query Processing: Main Steps III

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.
6. Application of methods defined in the clause SELECT to the suitable objects of the final candidate solutions.



The Clinical Database I

- 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.
- These patients suffer from an insufficient supply of blood to the coronary arteries due to a partial (or total) obstruction of some coronary vessels.
- Coronary revascularization is performed by inflations of a balloon, placed on a suitable catheter (**PTCA - Percoutaneous Transluminal Coronary Angioplasty**).

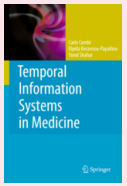


The Clinical Database II

- 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:

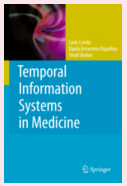
- *patient ID data*;
- *auxiliary demographic data*;
- *data related to risk factors*;
- *data related to current and previous therapies*;
- *data related to current and previous diagnoses*;
- *data related to follow-up visits*.



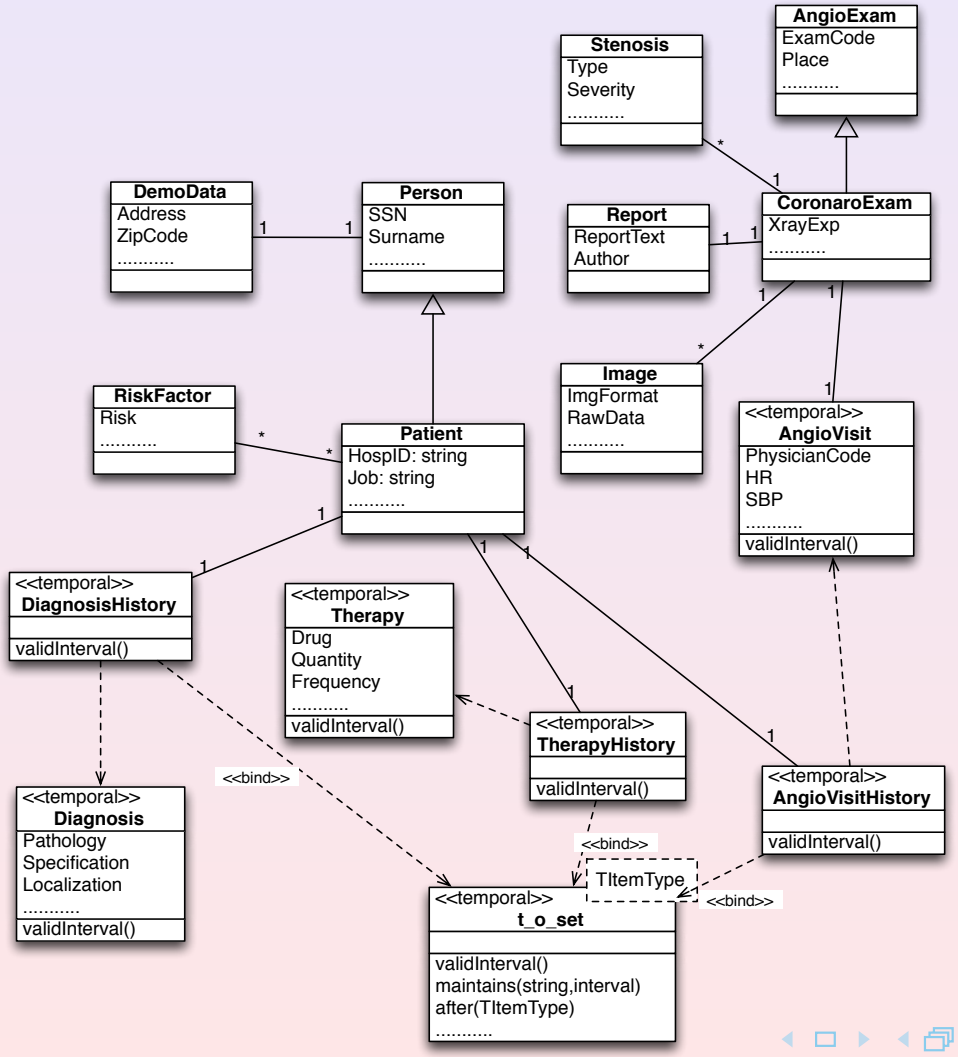
The Clinical Database III

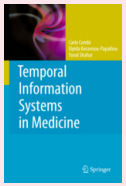
The clinical database contains some temporal objects, modeled through the temporal types:

- `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 Clinical Database IV

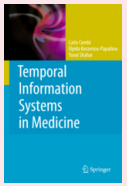




The Clinical Database: query example I

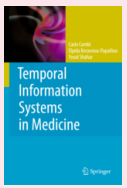
Example

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.



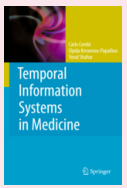
The Clinical Database: query example II

```
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
```

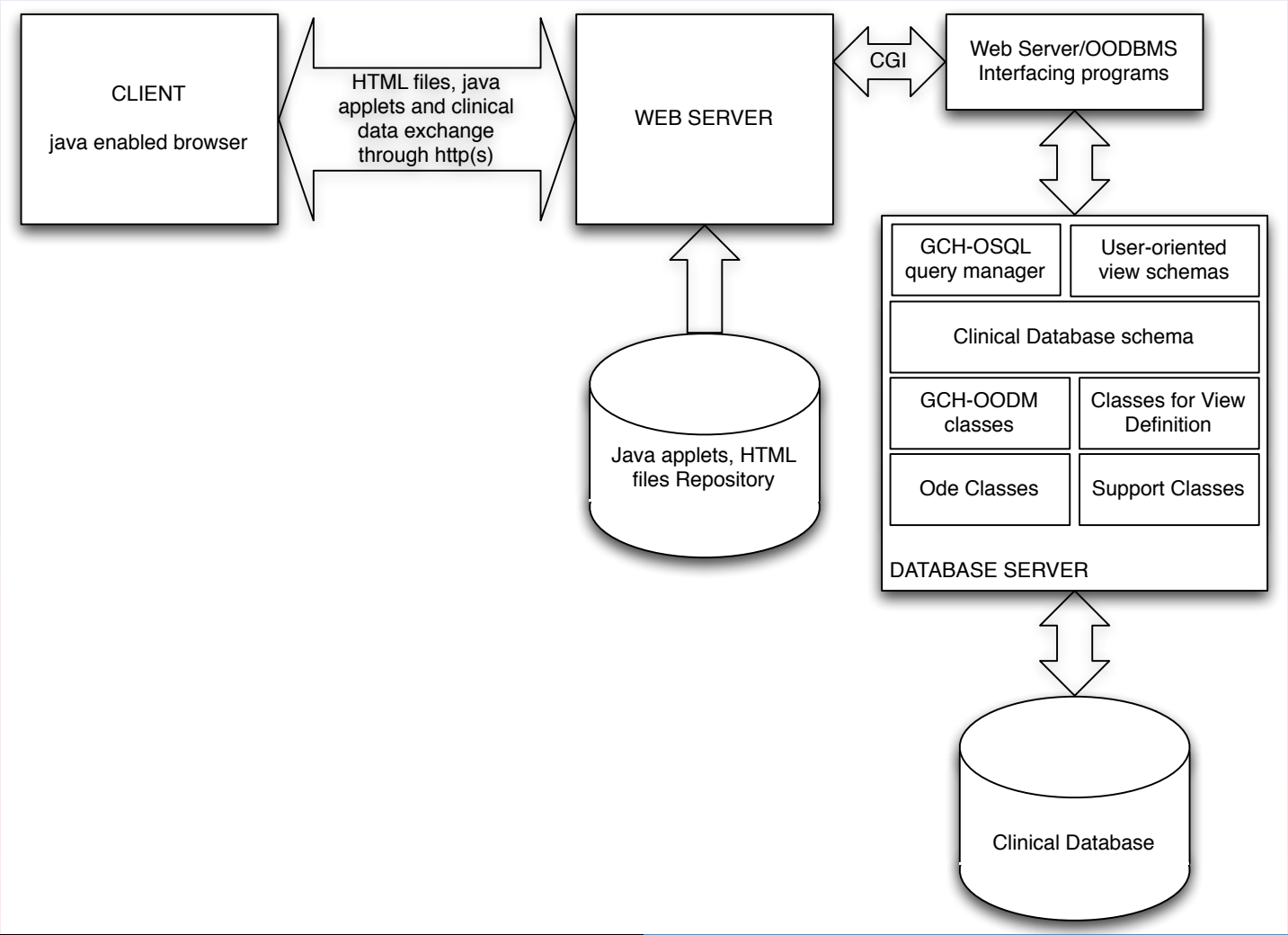


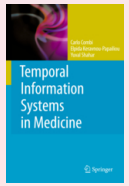
System Implementation

- GCH-OODM and GCH-OSQL are characterized by the presence of some prototypical implementations.
- Initially, a prototype of GCH-OSQL, based on the ONTOS OO DBMS, and a graphical interface have been implemented on a Sun workstation, in the OpenLook graphical environment.
- A second prototype system was designed and implemented using the OODBMS Ode.
- Finally, the Ode-based prototype has been extended to allow a web-based interaction with the system: **KHOSPAD (Knocking at the Hospital for PAtient Data)**.

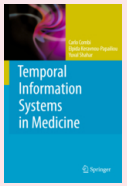


KHOSPAD Architecture



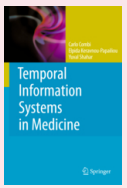


Outline

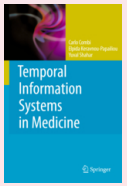


Further Research Directions

- Adoption of advanced data models.
- Maintenance of clinical raw data and abstractions.
- Merging the functions of temporal reasoning and temporal maintenance.
- Resolution of conflicts between temporal-reasoning and temporal data-base systems within hybrid architectures.
- Providing efficient storage protocols for hybrid architectures.
- Temporal clinical data warehousing and mining.
- Extraction of temporal information from unstructured clinical data.



Outline



Take Home Summary

- In this chapter we have overviewed some domain-specific, yet of general interest, topics related to the management, modeling and querying of temporal clinical data.
  - multiple temporal dimensions of clinical data
    - relational approach
  - multiple granularities and indeterminacy in modeling and querying clinical data
    - relational approach
    - object-oriented approach