Modeling Medical Trials in Pharmacoeconomics using a Temporal Object Model

Iqbal A. Goralwalla, M. Tamer Özsu and Duane Szafron
Laboratory for Database Systems Research
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2H1
{iqbal,ozsu,duane}@cs.ualberta.ca

Abstract

Time is an inherent feature of many medical applications. These applications can also benefit from the support of object database management systems which better capture the semantics of the complex objects that arise in the medical domain. In this paper, we present a uniform behavioral temporal object model which includes a rich and extensible set of types and behaviors to support the various features of a medical application. We concentrate here on the application of pharmacoeconomic medical trials. Pharmacoeconomics is a field of medical economics in which the costs and outcomes of alternative treatments are assessed and compared, in order to establish which is the most appropriate treatment for a particular illness in a particular setting. We describe in detail the histories and timelines features of our temporal model and show how they can effectively be used to model a pharmacoeconomic trial. We then give an instance of a pharmacoeconomic trial as it would appear in the temporal object model and show, using queries, how a series of different behaviors could be used to retrieve various components of the instance. These components could then be used to assess the alternative treatments involved in the trial and determine their cost-effectiveness.

Keywords: medicine, medical trial, object models, pharmacoeconomics, temporal databases

1 Introduction

Most of the research on temporal databases has concentrated on extending the relational model [4] to handle time in an appropriate manner. However, it is now widely recognized that the relational model is inadequate for capturing the semantics of the complex objects that arise in many application domains. This has led to research into next-generation data models, specifically

object database management systems (ODBMSs). In such systems, we can more accurately capture the semantics of complex objects and treat time as a basic component versus an additional attribute as is the case in the relational systems.

Most of the medical applications for which ODBMSs are expected to provide support exhibit some form of temporality. Some examples are the following:

- *Nuclear medicine image analysis* – dynamic studies of the renal function are carried out in which the rate at which a radio opaque substance in excreted by the kidney is assessed with a series of X rays taken at particular times.

- *Medical records of patients* – this includes their surgical history, gynecological history, family history, drug history, etc.

- *Pharmacoeconomics* – in which the costs and outcomes of alternative treatments, administered over a period of time, are assessed and compared, in order to establish the most appropriate treatment for a particular illness.

- *Antenatal care* – the progress of a woman's pregnancy is mapped (followed) at a series of clinics attended from early to late pregnancy (in order to ensure the best outcome to the pregnancy). The information collected at each visit may include blood pressure, growth of fetus, hematology tests, drug history, etc.

- *Multidisciplinary diabetic care clinics* – since diabetes is a disease which can affect many of the body's systems, diabetic care necessarily involves many medical specialties. Multidisciplinary diabetic care clinics are set up so that all aspects of the patient's health can be followed and treated as necessary, at one clinic. Information collected at each clinic visit may include a hematology blood test, a record of home sugar testing, blood pressure, a photograph of the retina, etc.

Thus, time is an inherent feature of medical applications that require the functionality of ODBMSs. In this paper we describe a temporal object model which is sufficiently powerful to meet the requirements of these applications. We specifically show how the temporal object model is used to store and retrieve historical information in pharmacoeconomic medical trials[1]. Our work is

[1] Although we concentrate in this paper on medical trials in the field of pharmacoeconomics, our work can be applied in general to any medical or clinical trial.

conducted within the context of the TIGUKAT[2] system [21], briefly described in Section 2.

In the next sub-section, we describe a pharmacoeconomics study which motivates the features that our temporal model provides. We show, in the rest of the paper, how the various features of the temporal model can be used to represent, store, analyze, and reason about the components of the study.

1.1 Medical Trials in Pharmacoeconomics

One of the methods used in the field of pharmacoeconomics is cost-effectiveness analysis of different treatments [16]. This method has been used in the pharmacoeconomic analysis of treatments for illnesses such as pneumonia, asthma, chest infection, etc. In these trials, the group of patients suffering from the illness of interest are divided into sub-groups. Each of these sub-groups is administered one of the different treatments under investigation. Pharmacoeconomic analysis has so far mainly focussed on the comparison of different drugs used for treating the same illness. During the course of the trial, an object database of information is maintained which will be used in the cost-effectiveness analysis of each treatment. To illustrate, consider a trial comparing two different antibiotic treatments. The information required would include:

- *Antibiotic Related Costs* – The different antibiotics used and the time period during which they were applied. For each antibiotic the costs related to its use are also recorded. These include:
  - Acquisition cost
  - Preparation and administration costs
  - Laboratory monitoring costs
  - Cost of treating adverse effects
- *Infection Related Costs* – These include:
  - Laboratory monitoring tests (for example, microbiology blood tests and radiology tests), the times the patient took the tests, and the cost incurred for each test
  - Health care used (this includes emergency room visits, hospital bed costs, etc.) and related cost

[2]TIGUKAT (tee-goo-kat) is a term in the language of Canadian Inuit people meaning "objects." The Canadian Inuits (Eskimos) are native to Canada with an ancestry originating in the Arctic regions.

In addition to this information, a medical trial has various temporal modeling requirements. These include:

- A branching model of time in which histories of alternate treatments of a medical trial could be represented, and subsequently analyzed for their cost-effectiveness.
- The different kinds of medication and the time periods during which they were administered during the course of a particular treatment in the medical trial.
- The time periods during which different treatments in the medical trial were administered.
- Whether a patient in the current medical trial underwent a similar medical trial in the past.
- The different blood tests a patient took while undergoing a particular treatment.

We will show in the remainder of the paper how a medical trial can effectively be modeled in a temporal ODBMS. Analysts working in pharmacoeconomics can then use the ODBMS to retrieve components relevant to their line of investigation to aid their pharmacoeconomic analyses.

1.2 Organization of the paper

The rest of the paper is organized as follows. In Section 2, we briefly describe the TIGUKAT object model. In Section 3 we describe the histories and timelines features of our temporal model and show how they can be used in modeling the components of a medical trial as described in Section 1.1. We describe the type system for the medical trial object database in Section 4. In Section 5 we compare our model with other temporal object models. Section 6 concludes the paper and outlines future research.

2 Object Model Overview

The TIGUKAT object model [22, 21] is purely *behavioral* with a *uniform* object semantics. The model is *behavioral* in the sense that all access and manipulation of objects is based on the application of behaviors to objects. The model is *uniform* in that every component of information, including its semantics, is modeled as a *first-class object* with well-defined behavior. Other typical object modeling features supported by TIGUKAT include strong object identity, abstract types, strong typing, complex objects, full encapsulation, multiple inheritance, and parametric types.

The primitive objects of the model include: *atomic entities* (reals, integers, strings, etc.); *types* for defining common features of objects; *behaviors* for specifying the semantics of operations that may be performed on objects; *functions* for specifying implementations of behaviors over types; *classes* for automatic classification of objects based on type[3]; and *collections* for supporting general heterogeneous groupings of objects. In this paper, a reference prefixed by "T_" refers to a type, "**C_**" to a class, "*B_*" to a behavior, and "T_X< T_Y >" to the type T_X parameterized by the type T_Y. For example, T_person refers to a type, **C_person** to its class, *B_age* to one of its behaviors and T_collection< T_person > to the type of collections of persons. A reference such as David, without a prefix, denotes some other application specific reference.

The access and manipulation of an object's state occurs exclusively through the application of behaviors. We clearly separate the definition of a behavior from its possible implementations (functions). The benefit of this approach is that common behaviors over different types can have a different implementation in each of the types. This provides direct support for behavior *overloading* and *late binding* of functions (implementations) to behaviors.

The model separates the definition of object characteristics (a *type*) from the mechanism for maintaining instances of a particular type (a *class*). A *type* defines behaviors and encapsulates behavior implementations and state representation for objects created using that type as a template. The behaviors defined by a type describe the *interface* to the objects of that type. A *class* ties together the notions of *type* and *object instances*. Objects of a particular type cannot exist without an associated class and every class is uniquely associated with a single type. Object creation occurs only through classes using its associated type as a template for the creation. Thus, a fundamental notion of TIGUKAT is that *objects* imply *classes* which imply *types*.

In addition to classes, a *collection* is defined as a general grouping construct. It is similar to a *class* in that it groups objects, but it differs in some respects. First, object creation cannot occur through a collection, only through classes. Second, an object may exist in any number of collections, but is a member of the shallow extent of only one class. Third, classes are automatically managed by the system based on the subtype lattice whereas the management of collections is *explicit*, meaning the user is responsible for their extents.

[3]Types and their extents are separate constructs in TIGUKAT.

3 The Temporal Object Model

Our temporal object model includes a rich and extensible set of types and behaviors to support various notions of time. This section contains a a brief overview of the temporal primitives supported in the model, followed by a detailed description of how historical information and different types of timelines are modeled in TIGUKAT. We do not discuss other features of the temporal model, viz temporal domains, temporal granularities, calendars, and temporal indeterminacy here. We refer the reader to [13, 14] for a more detailed description of these features.

3.1 Temporal Ontology

We identify a *time interval* as the basic anchored specification of time and provide a wide range of operations on time intervals. The `T_interval` type is introduced to model time intervals. Unary operators are defined which return the lower bound, upper bound, length, and timeline of a time interval. A rich set of ordering operations among intervals is also defined [1], e.g., *precedes*, *overlaps*, *during*, etc. Set-theoretic operations viz *union*, *intersection* and *difference* are also defined on time intervals[4]. A time duration can be added or subtracted from a time interval to return another time interval. A time interval can be expanded or shrunk by a specified time duration. We model different kinds of open, closed, half open and half closed intervals. The `T_interval` type is introduced to model time intervals.

A *time instant* (*moment*, *chronon*, etc.) is a specific anchored moment in time. We model a time instant as a special case of a (closed) time interval which has the same lower and upper bound, e.g., $January\ 24 = [January\ 24, January\ 24]$. A wide range of operations can be performed on time instants. A time instant can be compared with another time instant with the transitive comparison operators $<$ and $>$. A time duration can be added to or subtracted from a time instant to return another time instant. A time instant can be compared with a time interval to check if it falls before, within or after the time interval.

We identify a *time span* as being an unanchored relative duration of time. A time span is basically an atomic cardinal quantity, independent of any time instant or time interval. Time spans have a number of operations defined on them. A time span can be compared with another

[4]Note that the union of two disjoint intervals is not normally regarded as an interval. Similarly, for the difference operation, if the second interval is contained in the first, the result is not an interval. In our model, these cases are handled by returning an object of the *null* type (`T_null`). The `T_null` type is a subtype of all other types in the TIGUKAT type lattice, including the *interval* type (`T_interval`). Hence, every instance of `T_null` is also an instance of `T_interval`.

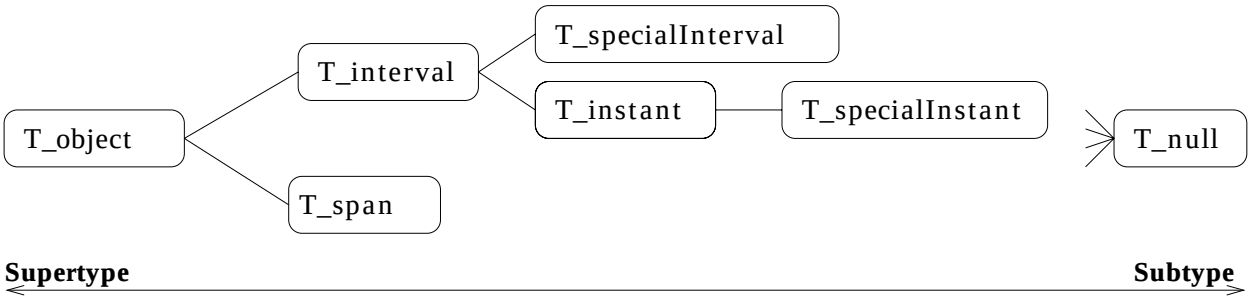

Figure 1: The temporal ontology type hierarchy.

time span using the transitive comparison operators $<$ and $>$. A time span can be subtracted from or added to another time span to return a third time span.

Figure 1 shows the type hierarchy of the types used to model time intervals, time instants, and time spans. We define the `T_specialInterval` type as a subtype of `T_interval`. The `T_specialInterval` type has only one instance which represents the *null* interval[5]. Since a time instant is a special case of a time interval in our model, the `T_instant` type is naturally defined as a subtype of `T_interval` to model time instants. There is also one additional instant type `T_specialInstant` defined as a subtype of `T_instant`. The only instances of this type are $-\infty$ and $+\infty$. These instances are used as minimum and maximum bounds of a timeline as will be discussed in Section 3.3. Finally, the type `T_span` models time spans. Detailed signatures of the `T_interval`, `T_instant`, and `T_span` types are given in the Appendix.

3.2 Temporal Histories

One requirement of a temporal model is an ability to adequately represent and manage histories of real-world events and histories of objects in the object database. In this section we first describe the types and behaviors in TIGUKAT that can be used to model histories of real-world events. Building on these types and behaviors, we then show how object histories are uniformly modeled in TIGUKAT.

3.2.1 Real-World Event Histories

Our model represents the temporal histories of real-world objects whose type is `T_X` as objects of the `T_history<T_X>` type. A temporal history consists of objects and their associated timestamps.

[5]Null intervals are usually a result of set-theoretic operations. For example, in the set difference operation, if the first interval is contained in the second, a null interval is returned.

One way of modeling a temporal history would be to define a behavior which returns a collection of <timestamp, object> pairs. However, instead of structurally representing a temporal history in this manner, we use a behavioral approach by defining the notion of a *timestamped object*. A timestamped object knows its timestamp and its associated value at (during) the timestamp. A temporal history is made up of such objects. The following behaviors are defined on the `T_history<T_X>` type:

$$
\begin{aligned}
B\_history: &\quad \texttt{T\_collection} < \texttt{T\_timeStampedObject} < \texttt{T\_X} >> \\
B\_timeline: &\quad \texttt{T\_timeline} \\
B\_insert: &\quad \texttt{T\_X}, \texttt{T\_interval} \rightarrow \\
B\_validObjects: &\quad \texttt{T\_interval} \rightarrow \texttt{T\_collection} < \texttt{T\_timeStampedObject} < \texttt{T\_X} >>
\end{aligned}
$$

Behavior *B_history* returns the set (collection) of all timestamped objects that comprise the history. A history object also knows the timeline it is associated with and this timeline is returned by the behavior *B_timeline*. The timeline basically orders the time intervals (or time instants) of timestamped objects. Timelines are described in Section 3.3. Another behavior defined on history objects, *B_insert*, uses an object and a time interval to create a timestamped object and inserts it into the history. The *B_validObjects* behavior allows the user to get the objects in the history that were valid at (during) a given time interval.

Each timestamped object is an instance of the `T_timeStampedObject<T_X>` type. This type represents objects and their corresponding timestamps. Behaviors *B_value* and *B_timeStamp* defined on `T_timeStampedObject` return the value and the timestamp (time interval) of a timestamped object, respectively.

$$
\begin{aligned}
B\_value: &\quad \texttt{T\_X} \\
B\_timeStamp: &\quad \texttt{T\_interval}
\end{aligned}
$$

Example 3.1 It is desirable to represent a patient's blood test history over the course of a particular illness. Each blood test is represented by an object of the type `T_bloodTest`. Therefore, the history of the patient's blood tests is represented by an object of type `T_history<T_bloodTest>`. Let us call this object bloodTestHistory. Now, suppose the patient was suspected of having septicemia, and therefore, had diagnostic hematology and microbiology blood tests on January 15, 1995. As a result of a very raised white cell count the patient was given a course of antibiotics while the results of the tests were pending. A repeat hematology test was ordered on February 20,

1995. To record these tests, three objects with type `T_bloodTest` were created and then entered into the object database using the following TIGUKAT behavior applications:

$$\begin{aligned}
&\mathsf{bloodTestHistory}.B_insert(\mathsf{microbiology}, January\ 15,\ 1995)\\
&\mathsf{bloodTestHistory}.B_insert(\mathsf{hematology1}, January\ 15,\ 1995)\\
&\mathsf{bloodTestHistory}.B_insert(\mathsf{hematology2}, February\ 20,\ 1995)
\end{aligned}$$

If subsequently there is a need to determine which blood tests the patient took in January 1995, this would be accomplished by the following behavior application:

$$\mathsf{bloodTestHistory}.B\_validObjects([January\ 1,\ 1995,\ January\ 31,\ 1995])$$

This would return a collection of the two timestamped objects, {timeStampedMicrobiology, timeStampedHematology1}, representing the blood tests the patient took in January 1995. The first timestamped object would have microbiology as its value and the second would have hematology1 as its value[6].

To assist in clarifying the contents and structure of a history object, we give a pictorial representation of bloodTestHistory in Figure 2. In the figure, the boxes shaded in grey are objects. Objects have an outgoing edge labeled by each applicable behavior, and leading to the object resulting from the application of the behavior. For example, applying the behavior *B_timeline* to the object bloodTestHistory results in the object bloodTestTimeline. A circle labeled with the symbols { } represents a collection object and has outgoing edges labeled with "$\in$" to each member of the collection. For example, applying the *B_history* behavior to the object bloodTestHistory results in a collection object whose members are the timestamped objects timeStampedMicrobiology, timeStampedHematology1, and timeStampedHematology2. Finally, the *B_insert* and *B_validObjects* behaviors for bloodTestHistory are defined and point to function objects. The *B_insert*(bloodTestHistory) function object updates the blood test history when given an object of type `T_bloodTest` and a time interval. Similarly, the *B_validObjects*(bloodTestHistory) function object returns a collection of timestamped objects when given a time interval. □

In this section we illustrated how history objects can be used to represent the histories of real events. Another usage of history objects would be to represent the valid time and transaction time

[6]It should be noted that although we have two different timestamped objects containing the values microbiology and hematology1, they both contain the same time instant. That is, although timeStampedMicrobiology.*B_value* = microbiology and timeStampedHematology1.*B_value* = hematology1, timeStampedMicrobiology.*B_timestamp* = timeStampedHematology1.*B_timestamp* = *January* 15, 1995.

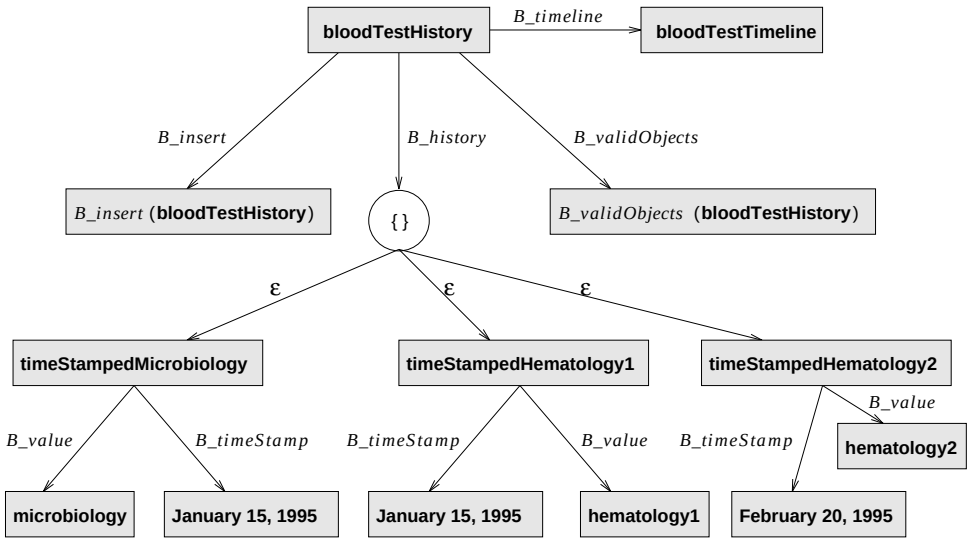

Figure 2: A pictorial representation of the bloodTestHistory object.

histories of the objects present in the database. We now describe how we manage these kinds of histories.

3.2.2 Valid and Transaction Time Histories

To represent histories of objects in the object database, we associate temporality with class objects. To manage temporal information of objects, the type `T_temporalClass<T_X>` is introduced as a subtype of the primitive type `T_class<T_X>`. An instance of `T_temporalClass<T_X>` (or any of its subtypes) is called a *temporal class*, and so objects belonging to a temporal class are called *temporal objects*.

TIGUKAT treats everything as objects; consequently classes are objects too. It therefore makes sense to distinguish between the notions of *class* temporality and *object* temporality. An object that is not a class can be either temporal or non-temporal whereas a class object may be temporal as a class or temporal as an object[7].

Definition 3.1 *Temporality of Objects:* An object o is temporal as an object if and only if the class of o is a temporal class. ■

Definition 3.2 *Temporality of Classes:* A class object o is temporal as a class if and only if its type is a subtype of `T_temporalClass`. ■

[7] We choose to use the same word (*temporal*) for both these notions since it is usually clear from the context which one of the two we are talking about. In the rest of the paper, unless otherwise specified, *temporal class* will mean the instances of the class are temporal.

These definitions imply that temporality of objects in TIGUKAT is not orthogonal to their class. If a class is temporal, then all of its members are temporal; if a class is non-temporal, then none of its members is temporal.

Type `T_temporalClass` defines additional functionality for representing the semantics of the temporality of objects. It allows its instances (which are temporal classes) to maintain histories of their constituent objects. A temporal class also maintains a history of when updates to the object were recorded in the object database. The former is known as the *valid time* history and the latter as *transaction time* history [27]. It is well known that these are orthogonal histories. The `T_temporalClass` type defines two behaviors to model valid time and transaction time histories independently of one another as follows:

$$B\_validHistory: \quad \texttt{T\_history} < \texttt{T\_X} >$$
$$B\_transHistory: \quad \texttt{T\_history} < \texttt{T\_X} >$$

Behaviors *B_validHistory* and *B_transHistory* return the valid and transaction time history of an object belonging to the type `T_X`, respectively. Our approach of separating valid and transaction times into two behaviors is in contrast to [24, 12] where they are structurally modeled together. This usually requires an object value to have corresponding entries for *both* the valid and transaction times. The separation in modeling of valid and transaction times is more intuitive and gives substantial flexibility as it allows different types of object databases to be defined as per application needs [26]:

- *Rollback* or *transaction-time* object databases can be modeled using the *B_transHistory* behavior. This facilitates the state of the object database to be seen *as of* some particular time.

- *Historical* object databases can be modeled using the *B_validHistory* behavior. This shows the time when the stored historical information was valid.

- *Temporal* object databases encompass the functionalities of rollback and historical object databases and are, therefore, modeled using both the *B_validHistory* and *B_transHistory* behaviors.

Furthermore, other dimensions of time (e.g., event time [24], user-defined time) can easily be modeled by adding the corresponding behaviors in the interface of the `T_temporalClass`:

$$
\begin{aligned}
B\_eventHistory: &\quad \texttt{T\_history} < \texttt{T\_X} > \\
B\_userDefinedHistory: &\quad \texttt{T\_history} < \texttt{T\_X} > \\
&\vdots
\end{aligned}
$$

This shows the extensibility and uniformity of our approach in modeling histories of objects in the object database.

Example 3.2 Suppose the hospital staff wants to maintain a record of the time when operations in the hospital took place. In this case, the class of all operations (**C_operation**) will be a temporal class with type `T_temporalClass<T_operation>`. **C_operation** will then consist of temporal objects having the semantics of different operations and the times they took place. The following behavior returns the history of different operations that took place in the hospital:

$$\textbf{C\_operation}.B\_validHistory$$

Applying the behavior *B_history* to this object returns a collection consisting of timestamped operations. If we were also interested in the times when operations in the hospital were entered in the object database, then we would simply use the *B_transHistory* behavior to access these. □

Figure 3 shows the type hierarchy of the history types described in this section. In the next section, we introduce the concept of a timeline and show how it is used to give an ordering to the timestamps in a temporal history.

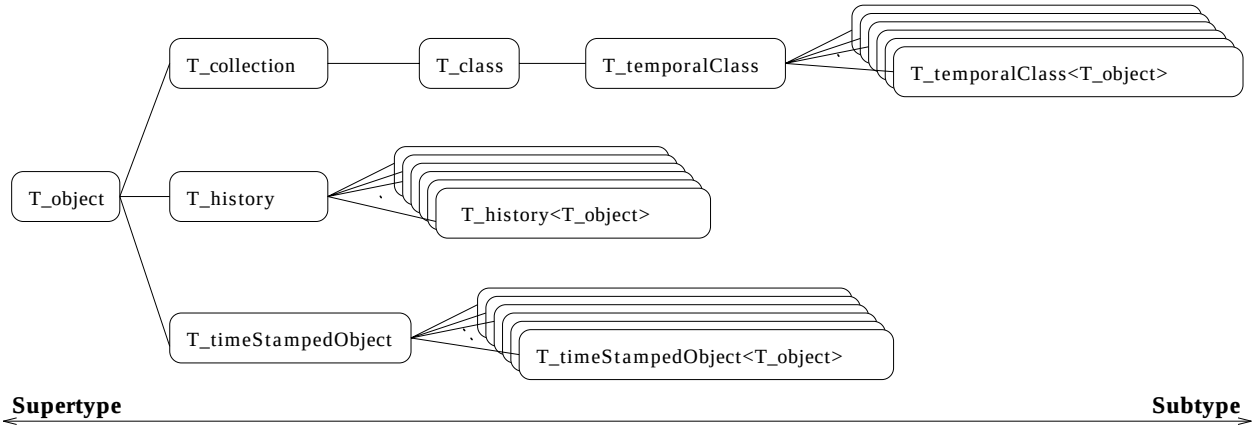

Figure 3: The temporal histories type hierarchy.

3.3 Timelines

In our model, a *timeline* represents an axis over which time can be perceived in an ordered manner. Basically, timelines are used to give an order to the timestamps in histories.

Definition 3.3 *Timeline ($\mathcal{T}_\mathcal{L}$):* A timeline $\mathcal{T}_\mathcal{L}$ is a triplet $\langle \mathcal{O}, \mathcal{I}, \mathbf{L\_T} \rangle$, where $\mathcal{O}$ is the temporal order of $\mathcal{T}_\mathcal{L}$, $\mathcal{I}$ is the defining time interval of $\mathcal{T}_\mathcal{L}$, and $\mathbf{L\_T}$ is a collection of the timestamps (time intervals) which belong to $\mathcal{T}_\mathcal{L}$. ■

A temporal order of a timeline can be classified as being *linear* or *branching*. In a linear order, time flows from past to future in an ordered manner. Since a timestamps on a timeline are basically time intervals, we distinguish between linear orders in which the time intervals can overlap and those in which the time intervals are strictly non-overlapping.

Definition 3.4 *Sub-linear Order:* Let t_i and t_j be anchored timestamps. Then,
$\forall t_i t_j (t_i \texttt{ overlaps } t_j \vee t_i \texttt{ precedes } t_j \vee t_j \texttt{ precedes } t_i)$ ■

Definition 3.5 *Linear Order:* Let t_i and t_j be anchored timestamps. Then,
$\forall t_i t_j (t_i \texttt{ precedes } t_j \vee t_j \texttt{ precedes } t_i \vee t_i \texttt{ equals } t_j)$ ■

In a sub-linear order, timestamps are allowed to overlap each other while in a linear (total) order, timestamps strictly follow or precede each other, i.e., they do not overlap.

In a branching order, time is linear in the past up to a certain point, at which it branches out into alternate futures.

Definition 3.6 *Branching Order:* Let t_i, t_j and t_k be anchored timestamps. Then,
$\forall t_i t_j t_k ((t_j \texttt{ precedes } t_i \wedge t_k \texttt{ precedes } t_i) \rightarrow (t_j \texttt{ precedes } t_k \vee t_j \texttt{ overlaps } t_k \vee t_k \texttt{ precedes } t_j))$ ■

The branching order defined above is a forward (in time) branching order. It ensures the two predecessors of a given time are comparable. The structure of a branching order can be thought of as a tree defining a partial order of times. The trunk (stem) of the tree is a linear order and each of its branches is a branching order. The branching order is useful in applications such as computer aided design and planning or version control which allow objects to evolve over a non-linear (branching) time dimension (e.g., multiple futures, or partially ordered design alternatives).

Without loss of generality, in the rest of the paper timelines with a linear order will be referred to as *linear timelines* while those with a branching order will be referred to as *branching timelines*.

To facilitate ordering on timelines, behaviors *B_prev* and *B_next* are defined on `T_interval` (see the Appendix) to return a collection of the previous or next time intervals of a time interval. For time intervals belonging to totally ordered linear timelines, the collection of the previous or next time intervals would be comprised of only one time interval since the time intervals in a totally ordered linear timeline are non-overlapping.

Definition 3.7 *Defining time interval ($\mathcal{I}$):* A defining time interval $\mathcal{I} = [t_s, t_e]$ is a time interval over which a timeline $\mathcal{T}_\mathcal{L}$ is defined. t_s is a time instant denoting the start time of $\mathcal{T}_\mathcal{L}$ and t_e is a time instant denoting the end time of $\mathcal{T}_\mathcal{L}$. ■

A timeline is comprised of one defining time interval $\mathcal{I}$ which essentially determines the size of a timeline. We define two constants (which are essentially instances of `T_specialInstant`) $-\infty$ and $+\infty$ to be the lower (t_s) and upper (t_e) bounds of the longest defining time interval time of which a timeline could be comprised.

A timeline may then contain any number of additional time intervals and time instants with the restriction that each of them lies within the defining time interval $\mathcal{I}$. Over the lifetime of the timeline, $\mathcal{I}$ remains the same, but other additional time intervals and time instants may be added, deleted or modified. For example, consider a timeline with an $\mathcal{I}$ $[08:00\ January\ 1\ 1993,\ +\infty)$. Time intervals or time instants (which could be modeling the history of a particular object) such as [*January* 15 1993, *February* 20 1995], *March* 27 1995, 00 : 00 : 03 *May* 25 1995 can now be added to the timeline. These time intervals and instants form the collection $\mathbf{L}_\mathcal{T}$.

Since $\mathbf{L}_\mathcal{T}$ can consist of time intervals or time instants or both, we can define both *homogeneous* and *heterogeneous* timelines. This provides additional flexibility in defining histories of various activities and objects because some activities occur at moments in time, while others occur over a period of time.

Figure 4 shows the type hierarchy of the types used to model the different kinds of timelines described in this section. The abstract type `T_timeline` is first defined as a supertype of all linear and branching timelines, and defines the following behaviors:

$$\begin{aligned} B\_definingTimeInterval: &\quad \texttt{T\_interval} \\ B\_timeIntervals: &\quad \texttt{T\_collection} < \texttt{T\_interval} > \end{aligned}$$

B_definingTimeInterval returns the defining time interval of a timeline, while *B_timeIntervals* returns a collection of time intervals that have a certain temporal order. We then model linear

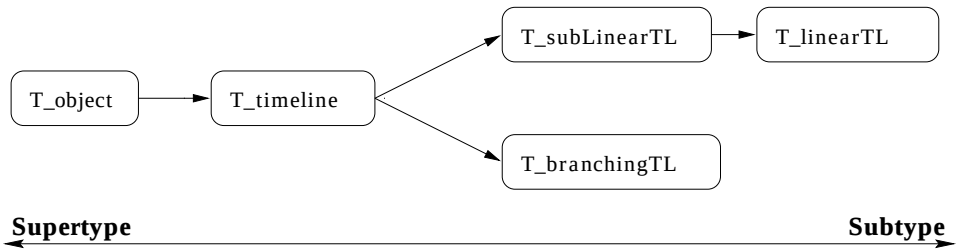

Figure 4: The timelines type hierarchy.

timelines based on the ordering[8] of the time intervals and time instants they contain. A sub-linear timeline is made up of time intervals which have a sub-linear order defined on them (i.e., the intervals are allowed to overlap). Type `T_subLinearTL` which is a subtype of `T_timeline` models these kind of timelines. Linear ordered timelines are made up of time intervals which have a linear order defined on them (i.e., the intervals are not allowed to overlap). These kind of timelines are modeled by the type `T_linearTL`. Since a linear order is also a sub-linear order, it follows that `T_linearTL` is a subtype of `T_subLinearTL`. The `T_subLinearTL` inherits the *B_definingTimeInterval* and *B_timeIntervals* behaviors from `T_timeline`. *B_timeIntervals* returns a collection of time intervals that have a sub-linear order defined on them. `T_subLinearTL` additionally defines a new behavior, *B_branchingTL* which returns the branching timeline to which the linear timeline belongs. If the linear timeline does not belong to any branching timeline, *B_branchingTL* returns an object of type `T_null`. To show the usefulness of the different kinds of linear timelines defined in our temporal model, we now consider several medical examples dealing with histories.

Example 3.3 Consider the operations that take place in a hospital on any particular day. It is usually the case that any given time multiple operations are taking place. Lets assume an eye cataract surgery took place between 8am and 10am, a brain tumor surgery took place between 9am and 12pm, and an open heart surgery surgery took place between 7am and 2pm on a certain day. Since different operations belong to the **C_operation** class (see Example 3.2), it is natural for them to share the same timeline. Figure 5 shows such a timeline, called operationsTimeline. As seen in the figure, the operations timeline consists of intervals (representing the time periods during which the different surgeries took place) that overlap each other. Hence, operationsTimeline is an example of a linear timeline which consists of time intervals that have a sub-linear order defined over them. □

[8]The ordering on timelines is defined by *B_precedes* and *B_overlaps* in `T_interval`.

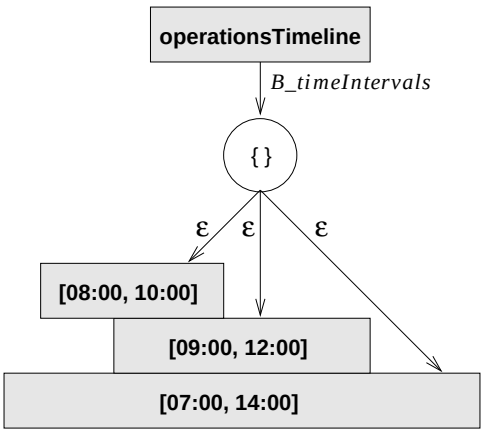

Figure 5: A pictorial representation of the operationsTimeline object showing its sub-linear order.

Example 3.4 To illustrate the use of totally ordered linear timelines, let us consider a patient with multiple pathologies, for example as a result of diabetes. The patient has to attend several special clinics, each on a different day. Hence, it follows that since the patient cannot attend more than one special clinic on any day, the timeline of the patient's special clinics history is linear and totally ordered. Suppose the patient visited the opthalmology clinic on January 10, 1995, the cardiology clinic on January 12, 1995, and the neurology clinic on February 3, 1995. Figure 6 shows a pictorial representation of specialClinicTimeline, which is the timeline for specialClinicHistory. As seen in Figure 6, specialClinicTimeline is linear and totally ordered as its time intervals do not overlap. □

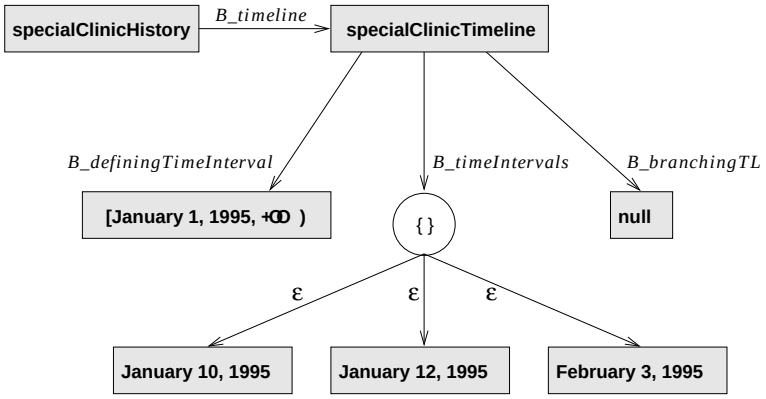

Figure 6: A pictorial representation showing the total order of the specialClinicTimeline linear timeline.

We now consider branching timelines. The `T_branchingTL` type models branching timelines and has the following new behaviors:

B_root : T_subLinearTL

$B\_branches$: T_collection < T_branchingTL >

B_in : T_interval → T_boolean

B_root returns the root of the branching timeline which is a sub-linear (or linear) timeline. $B\_branches$ returns a collection of branching timelines that form the branches of the main branching timeline. B_in checks whether a time interval is within the branching timeline. Branching timelines are most useful when employed to model versioning of some kind or in designing partially ordered alternatives.

Example 3.5 Consider an observational pharmacoeconomic analysis of the changing trends in the treatment of a chronic illness such as asthma. The analysis would be performed using information gathered over a time period. At a fixed point during this period new guidelines for the treatment of asthma were released. At that point the population of patients known to have asthma are divided into those whose doctors continue the old established treatment, and those whose doctors, in accordance with new recommendations, change their treatment. Thus, the patients are divided into two groups, each group undergoing a different treatment for the same illness. The costs and benefits accrued over the trial period for each treatment are calculated. Since such a study consists of several alternative treatments to an illness, a branching timeline is the natural choice for modeling the timeline of the study. The point of branching is the time when the new guidelines for the treatment of the illness are implemented. Figure 7 shows the branching timeline for such a medical trial history.

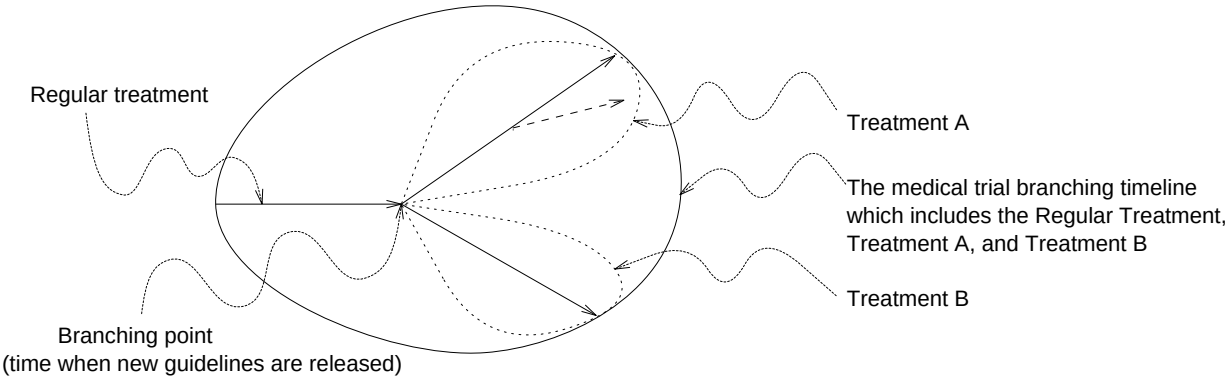

Figure 7: An example of a branching timeline.

The same branching timeline could as easily handle the situation when different versions of a particular treatment, say Treatment A, are implemented based on certain parameters. In this case, the "Treatment A" branch would in turn branch at a certain point into different Treatment A versions. This situation is also depicted in Figure 7. □

In the next section we show how the types and behaviors of histories and timelines can be used to model the medical trial described in Section 1.1.

4 The Medical Trial Object Database

4.1 Medical Trial Types and Behaviors

In this section we describe all the necessary types and behaviors which are needed to model a pharmacoeconomic trial (such as that describe in Section 1.1) in our temporal object model. Figure 8 shows the types used to model the various components of a medical trial. The behaviors of the types are given in Table 1.

We start by introducing the `T_medicalTrial` type to represent different alternative treatments in a medical trial. For example, treatmentA and treatmentB described in Example 3.5 are objects of type `T_medicalTrial` and represent the different treatments used in a medical trial. Since a medical trial is comprised of alternate treatments which take place during a time period, its semantics is best captured by a branching timeline as we showed in Example 3.5. The *B_timeline* behavior defined on `T_medicalTrial` returns the timeline that is associated with the medical trial as is depicted in Figure 7. Each treatment then has the same timeline. A treatment also has a collection of patients and antibiotics. To model these, `T_medicalTrial` defines the behaviors *B_patients* and *B_antibiotics*, respectively. For a given treatment, the *B_treatmentPeriod* behavior returns the time period during which the treatment took place. The illness for which the medical trial is being undertaken is given by the behavior *B_illness*.

B_antibiotics returns a collection of timestamped antibiotics. Each member of such a collection has an associated timestamp which gives us the time period during which the antibiotic was administered. In addition to the timestamp, each member also has an antibiotic object whose type is `T_antibiotic`. This type defines several behaviors which returns the various costs (outlined in Section 1.1) associated with an antibiotic. These behaviors are shown in Table 1.

The type `T_patient` represents the patients undergoing the medical trial. In the course of a treatment, a patient takes various blood tests and radiology tests. These are represented by the

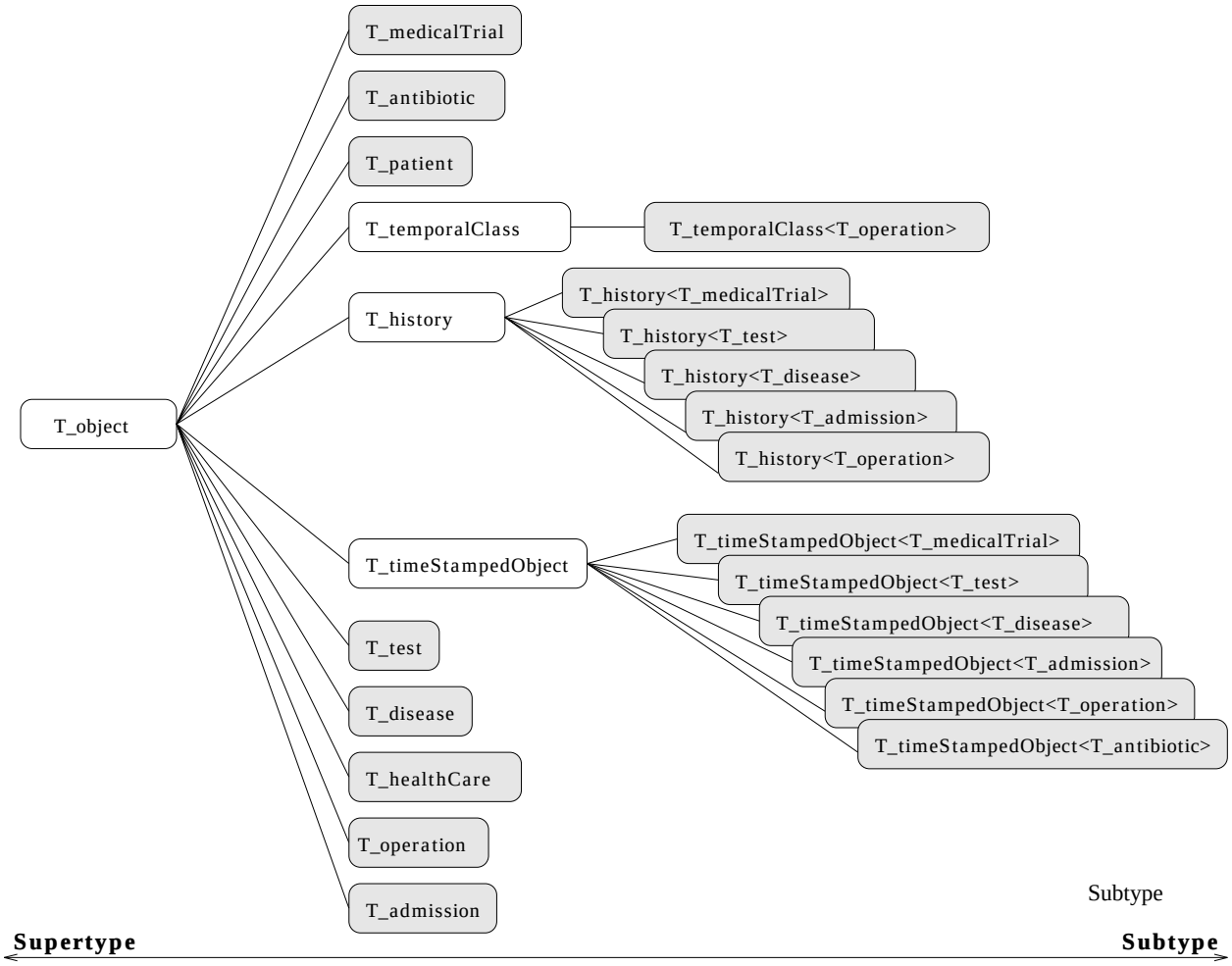

Figure 8: The type hierarchy for a medical trial.

type `T_test`. Behaviors *B_bloodTests* and *B_radiologyTests* are defined on `T_patient` to return the history of tests taken by a patient as shown in Table 1. Using histories to model a patient's tests enables us to look up the times when particular tests were taken. Type `T_test` defines the *B_cost* behavior which returns the cost associated with a particular treatment.

The behavior *B_diseases* defined on `T_patient` returns the disease history of a patient. It enables us to find out what diseases a patient has and at what times they were diagnosed. The *B_medicalTrials* behavior gives the history of all the different medical trials the patient has gone through up to date. Finally, the *B_healthCare* behavior returns the health care object of a patient.

The `T_healthCare` type represents the health care of a patient. Behavior *B_bedCost* returns the bed cost of a patient during the period of a given treatment of a medical trial. *B_emergencyVisits* returns the number of emergency visits of a patient during the period of a given treatment of a

Type	Signatures	
T_medicalTrial	*B_timeline*:	T_branchingTL
	B_antibiotics:	T_collection⟨T_timeStampedObject⟨T_antibiotic⟩⟩
	B_patients:	T_collection⟨T_patient⟩
	B_treatmentPeriod:	T_interval
	B_illness:	T_disease
T_antibiotic	*B_acquisitionCost*:	T_real
	B_prepAdmCost:	T_real
	B_labCost:	T_real
T_patient	*B_bloodTests*:	T_history⟨T_test⟩
	B_radiologyTests:	T_history⟨T_test⟩
	B_diseases:	T_history⟨T_disease⟩
	B_medicalTrials:	T_history⟨T_medicalTrial⟩
	B_healthCare:	T_healthCare
T_test	*B_cost*:	T_real
T_healthCare	*B_bedCost*:	T_medicalTrial → T_real
	B_emergencyVisits:	T_medicalTrial → T_integer
	B_admissions:	T_history⟨T_admission⟩
	B_operations:	T_history⟨T_operation⟩

Table 1: The medical trial types and behaviors.

medical trial. Finally, behaviors *B_admissions* and *B_operations* give the hospital admission and operations histories of a patient, respectively. Using histories to model a patient's hospital admissions enables us to easily determine when the patient was admitted and when he was discharged, and the number of times the patient has been admitted to the hospital. Similarly, modeling a patients operations using histories gives us the start and end times of each operation, and the number of operations.

4.2 A Medical Trial Instance

Having defined the types and behaviors used to model a medical trial, we show in Figure 9 a pictorial view of one of the treatments (treatmentA) in a medical trial. The pictorial view shows how, starting from treatmentA, various components of a treatment (as outlined in Section 1.1) can be reached using the behaviors defined in Table 1.

Figure 9 shows that treatmentA has a branching timeline medicalTrialBranchingTL which is shared by all other treatments. During the course of treatmentA, two antibiotics, antibioticA and antibioticB were used. antibioticA was administered during the time interval [January 1, 1995, April 1, 1995) and was followed by antibioticB which was administered for the time period [April 1, 1995, July 1, 1995). During the time period they were used, antibioticA incurred an acquisition cost of 45.0, a laboratory cost of 24.0, and a preparation and administration cost of 53.50, while antibioticB incurred costs of 55.0, 37.25, and 77.50, respectively.

patient1 was one of the patients that went through treatmentA. The previous medical trials

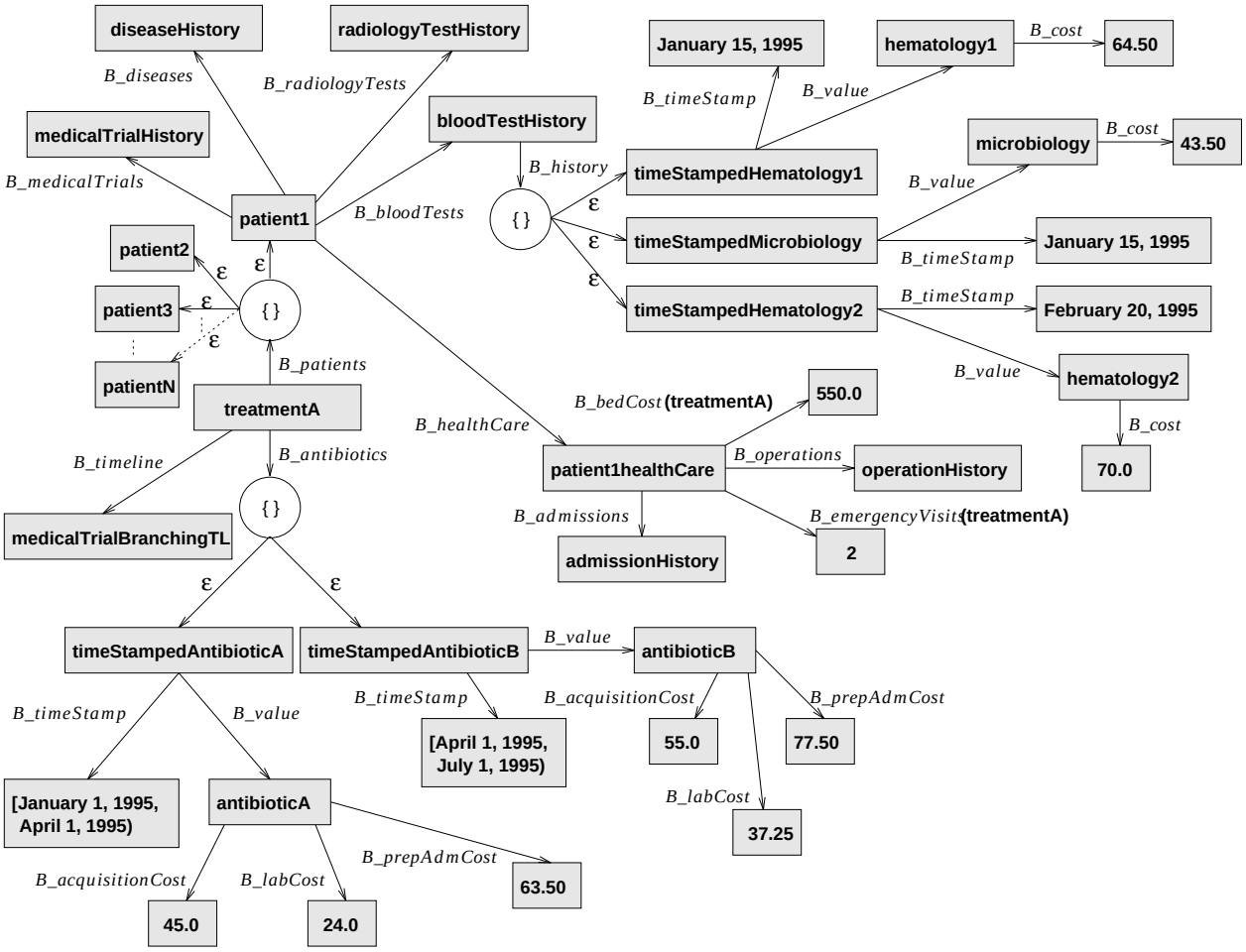

Figure 9: A pictorial representation of the components of a medical trial.

that patient1 went through are given by medicalTrialHistory. Similarly, patient1's history of diseases is given by diseaseHistory. The various monitoring tests taken by patient1 are given by radiologyTestHistory and bloodTestHistory. Figure 9 shows that patient1 had three blood tests done; hematology1 and microbiology were done on January 15, 1995 and incurred costs of 64.50 and 43.50, respectively. hematology2 was done on February 20, 1995 and incurred a cost of 70.0.

Finally, patient1HealthCare gives the health care of patient1. Using the hospital bed during the course of treatmentA incurred a cost of 550.0. patient1 had 2 visits to the emergency room while undergoing treatmentA. A history of patient1's hospital admissions is given by admissionHistory while the different operations he went through are given by operationHistory.

Researchers working on the pharmacoeconomic analysis of different treatments can now use the objects, types and behaviors shown in Figure 9, Figure 8 and Table 1 to retrieve costs of antibiotics, monitoring tests, hospital bed, etc. and the time period during which they occurred. They can then

use the retrieved information to carry out comparisons of the costs and effectiveness of alternative treatments. In the following section, we show how queries can be constructed in TIGUKAT to retrieve the various components of a medical trial.

4.3 Queries

4.3.1 The TIGUKAT Query Language

The TIGUKAT query model is a direct extension to the object model. It is defined by type and behavioral extensions to the primitive model. The languages for the query model include a complete calculus, and equivalent object algebra and a SQL-like user language [23]. In this section, we briefly discuss the TIGUKAT Query Language (TQL) and demonstrate how it can be used to access objects of a medical trial.

TQL is based on the SQL [11] paradigm and its semantics is defined in terms of the object calculus. Hence, every statement of the language corresponds to an equivalent object calculus expression. The basic query statement of TQL is the *select statement* which operates on a set of input collections, and returns a new collection as the result:

select < *object variable list* >
[**into** < *collection name* >]
  from < *range variable list* >
[**where** < *boolean formula* >]

The *select clause* in this statement identifies objects which are to be returned in a new collection. There can be one or more object variables with different formats (constant, variables, path expressions or index variables) in this clause. They correspond to free variables in object calculus formulas. The *into clause* declares a reference to a new collection. If the *into clause* is not specified, a new collection is created; however, there is no reference to it. The *from clause* declares the ranges of object variables in the *select* and *where* clauses. Every object variable can range over either an existing collection, or a collection returned as a result of a subquery, where a subquery can be either given explicitly, or as a reference to a query object. The *where clause* defines a boolean formula which must be satisfied by objects returned by a query. Two additional predicates are added to TQL boolean formulas to represent existential and universal quantification. The existential quantifier is expressed by the *exists predicate* which is *true* if the referenced collection is not empty. The universal quantifier is expressed by the *forAll predicate* which is *true* if for every element in every collection in the specified range variable list, the given boolean formula is satisfied.

4.3.2 Query Examples

To make the queries easier to read, we assume that treatmentA exists in the object database and is a member of the **C_medicalTrial** class. That is, we assume the presence of the "*treatmentA* **in C_medicalTrial**" construct in the *from clause* of each query.

Example 4.1 Which antibiotics were administered in treatment A and during what times?

select $timeStampedAntibiotic.B\_value$, $timeStampedAntibiotic.B\_timeStamp$

from $timeStampedAntibiotic$ **in** treatmentA.$B\_antibiotics$

This query simply goes through the collection of timestamped antibiotics of treatmentA and returns the antibiotic and timestamp associated with each timestamped antibiotic in the collection. As seen in Figure 9, this query would return a collection containing antibioticA, antibioticB and their associated timestamps (time intervals). □

Example 4.2 What was the total cost of using antibiotic A?

select $antibioticACost$

from $antibioticACost$ **in C_real**, $timeStampedAntibiotic$ **in** treatmentA.$B\_antibiotics$,

$antibioticA$ **in C_antibiotic**

where $(antibioticA = timeStampedAntibiotic.B\_value)$ **and**

$(antibioticACost = antibioticA.B\_acquisitionCost.B\_add(antibioticA.B\_labCost).B\_add(antibioticA.B\_p$

In this query, the acquisition, laboratory, and preparation and administration costs of antibiotic A are added to return the total cost, which would be 132.50 according to Figure 9. □

Example 4.3 Which antibiotics had an acquisition cost of more than $50?

select $timeStampedAntibiotic.B\_value$

from $timeStampedAntibiotic$ **in** treatmentA.$B\_antibiotics$

where $timeStampedAntibiotic.B\_value.B\_acquisitionCost > 50$

This query goes through the collection of timestamped antibiotics of treatmentA and returns the collection of antibiotics whose acquisition cost is greater than 50. From Figure 9, we see that a collection containing only antibioticB is returned as a result of this query. □

Example 4.4 Which blood tests did patient 1 take while antibiotic A was being administered, and what were the associated costs?

select $timeStampedbloodTest.B\_value, timeStampedbloodTest.B\_value.B\_cost$

from $patient1$ **in** treatmentA.$B\_patients$, $timeStampedbloodTest$ **in** $patient1.B\_bloodTests.B\_history$,

timeStampedAntibiotic **in** treatmentA.*B_antibiotics*

where (*antibioticA* = *timeStampedAntibiotic.B_value*) **and**

(*timeStampedbloodTest.B_timeStamp.B_within*(*timeStampedAntibiotic.B_timeStamp*))

This query returns patient1's blood tests (with their associated costs) whose timestamps fell within the time interval during which antibioticA was being used. From Figure 9, we see that these blood tests were hematology1, microbiology, and hematology2. □

Example 4.5 What was the time period during which treatment *A* took place?

select treatmentA.*B_treatmentPeriod*

This query returns a collection consisting of the time interval during which treatmentA took place. Assuming treatmentA ends when no more antibiotics are administered, an alternative approach would be to construct the query such that it goes through the collection of the timestamped antibiotics of treatmentA and takes the union of the timestamps associated with each antibiotic in the collection. Then, for the instance given in Figure 9, the query would return the time interval [January 1, 1995, July 1, 1995). □

Example 4.6 How many alternative treatments took place in the medical trial in which treatment *A* was administered?

select treatmentA.*B_timeline.B_branches.B_cardinality*

Since treatmentA has a branching timeline associated with it, the number of alternative treatments in the medical trial is simply the number of branches of the branching timeline. This is obtained by taking the cardinality of the collection returned as a result of the treatmentA.*B_timeline.B_branches* behavior application. If the medical trial consisted of treatment A and treatment B, then for the instance in Figure 9, this query returns a collection containing 2. □

Example 4.7 Has patient 1 undergone a medical trial for the same illness before?

select *timeStampedMedicalTrial.B_timeStamp*

from *patient*1 **in** treatmentA.*B_patients*, *timeStampedMedicalTrial* **in** *patient*1.*B_medicalTrials.B_history*

where *timeStampedMedicalTrial.B_value.B_illness.B_equal*(treatmentA.*B_illness*)

This query goes through patient1's medical trial history and checks if there exists an illness which is the same as the one for which treatmentA is being administered. □

Example 4.8 How many patients went through treatment *A*?

select treatmentA.*B_patients.B_cardinality*

The number of patients in treatmentA is the cardinality of the collection returned as a result of the treatmentA.*B_patients* behavior application. □

5 Related Work

There have been many object-oriented temporal model proposals (see, for example, [20, 24, 25, 17, 29, 12, 3, 28, 2]). These models differ in the functionality that they offer, but none of them provide support for all the requirements of a pharmacoeconomics medical trial. More specifically, in our model:

- Both linear and branching orders are supported. The above temporal ODBMSs support only linear time. As described in Section 1.1 and Example 3.5, a medical trial is composed of alternate treatments and hence needs a branching timeline. Furthermore, we support timelines comprised of time intervals or time instants or both. This facilitates the definition of both homogeneous and heterogeneous timelines, allowing us to model events that take place at particular moments of time and those that take place within a duration of time on the same timeline. To the best of our knowledge, this feature of heterogeneous timelines has not been addressed before in the temporal database research community.

- Since our model is behavioral, different dimensions of time (e.g., valid and transaction time dimensions) are represented using separate behaviors in contrast to structurally combining them in a single behavior. In other words, our approach is purely behavioral and encapsulates the structure within behaviors.

Among the temporal object models that have been proposed so far, [29] and [12] come closest to our work in modeling the various notions of time. However, these models leave a lot to the user to define as basic temporal support. We contend that specifying a schema for modeling histories and timelines in an application such as a medical trial which has diverse requirements, is not trivial. Hence in our temporal model, we provide a rich set of extensible types with corresponding behaviors. Using the time schema, we show how each of the components of a medical trial can uniformly and consistently be supported in our model.

There has been significant work on temporal clinical databases in the medical informatics field, ranging from temporal reasoning of clinical information to supporting multiple granularities in clinical data. Some proposals use the relational model as a basis for adding temporality, while

other proposals prefer the object-oriented model. We now describe some of the temporal clinical models which are related to the work presented in this paper.

Kahn and colleagues [19] describe a query language called TQuery that is designed specifically to formulate database queries that are dependent on temporal and contextual relationships. TQuery is used to retrieve patient data from an object-oriented patient medical-record system called the temporal network (TNET). TNET and TQuery support context-sensitive queries required by a rule-based expert system that reasons about therapy plans for a specific patient. In [18], TNET is described in more detail as a structure that provides a mechanism for marking important temporal events that occur during a patient's clinical course. Once created, a patient's TNET grows to reflect ongoing clinical events and serves as a temporal model of the patient's past medical history. An extension of TNET called ETNET is also described. ETNET supports both context-sensitive data storage and retrieval, and context-specific reasoning.

The work of Combi and colleagues [7, 6, 8, 9, 5] concentrates to a large extent on representing, storing and retrieving clinical data which is available in multiple granularities, in the context of object-oriented DBMSs. In [7] deficiencies are identified in the relational model in developing time-oriented medical record (TOMR) management systems. More specifically, the authors note the limitations of the relational model in supporting abstract data types and complex objects such as X-rays. Subsequently, they describe initial steps in designing and implementing a TOMR using an object-oriented DBMS. These steps include the definition of abstract data types in structuring clinical data and their temporal aspect, and implementation of granularities as related to temporal information. In [6] a graphical user-interface is designed and implemented. This user-interface displays temporal clinical information at different levels of granularity. The definition and implementation of an object-oriented model for representing temporal clinical data is discussed in [8, 9]. In [5] an object-oriented temporal extension of SQL, called GCH-OSQL, is described. GCH-OSQL allows storage of clinical information at different and mixed granularities, and also handles the uncertainty resulting from comparisons between temporal data of different granularities.

Das and colleagues [10] outline an algebra that maintains a consistent relational representation of temporal data, and allows different types of temporal queries needed for protocol-directed decision support. They then introduce Chronus, a temporal query system that can map between algebraic operations and SQL queries to relational databases. Their system has been applied to the task of screening patients for clinical trials. It can also be extended to model timestamps of varying granularities, and to insert and delete timestamped data.

6 Summary

In this paper we described the histories and timelines features of our temporal object model and showed how the history types and behaviors enable us to model the various histories of the components of a medical trial in the field of pharmacoeconomics. Furthermore, we outlined how any number of time dimensions (valid time, transaction time, etc.) can co-exist in our model. Valid and transaction time histories are modeled by separate behaviors thereby exactly modeling the conceptual notions of the two times. The concept of a timeline was then introduced and both linear and branching timelines were described. We showed how different histories in a medical trial need different kinds of timelines to order their timestamps.

Finally, appropriate types and behaviors were defined to represent the various components of a medical trial in our temporal object model. Using these types and behaviors, an instance of a medical trial was given. By making use of TQL and the instance, we illustrated how different series of behavior applications could be used to retrieve various objects of a medical trial. These objects could subsequently be used in the cost analysis of each treatment used in the medical trial.

A prototype implementation of the core TIGUKAT model is complete [15] and its temporal extension is currently under development. The completion of this task will allow us to apply the temporal object model to a real pharmacoeconomic medical trial. This will substantiate the practicality of the modeling aspects that have been proposed in this paper.

Our temporal object model also supports different temporal granularities [14] and temporal indeterminacy [13]. Usually, temporal information in medical applications is available in multiple granularities. For example, the history of the patient's blood tests (see Example 3.1) has a granularity of *days* (i.e., blood tests are taken on particular days) while the time periods of a patient's operations (see Example 3.2) are usually specified with a granularity of *minutes*. Temporal indeterminacy is also prevalent in medical applications. For example, if the condition of a patient is checked on an hourly basis and it was noticed that a patient's condition was significantly worse at a particular hour, say $6am\ May\ 30$, one can only reasonably conclude that his condition deteriorated sometime between $5:00am\ May\ 30$ and $5:59am\ May\ 30$. This further justifies the power of our temporal object model in supporting medical applications.

Acknowledgments

The idea behind this paper arose from discussions with Dr. Zahira Bachelani. We would like to thank her for her insight of the pharmacoeconomics field, and for her valuable comments which helped in the clarity and presentation of this paper. This work has been supported by the Natural Sciences and Engineering Research Council of Canada under research grants OGP0951 and OGP8191.

References

[1] J. F. Allen. Towards a General Theory of Action and Time. *Artificial Intelligence*, 23(123), 1984.

[2] T.S. Cheng and S.K. Gadia. An Object-Oriented Model for Temporal Databases. In *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, pages N1–N19, Arlington, Texas, June 1993. Also published as Technical Report TR-93-05.

[3] W.W. Chu, I.T. Ieong, R.K. Taira, and C.M. Breant. A Temporal Evolutionary Object-Oriented Data Model and Its Query Language for Medical Image Management. In *Proc. 18th Int'l Conf. on Very Large Data Bases*, pages 53–64, August 1992.

[4] E.F. Codd. A Relational Model for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, 1970.

[5] C. Combi, F. Pinciroli, M. Cavallaro, and M. Cucchi. Querying Temporal Clinical Databases with Different Time Granularities: The GCH-OSQL Language. In R.M. Gardner, editor, *19. Annual Symposium on Computer Applications in Medical Care*, pages 326–330. Hanley and Belfus, 1995.

[6] C. Combi, F. Pinciroli, G. Musazzi, and C. Ponti. Managing and Displaying Different Time Granularities of Clinical Information. In J.G. Ozbolt, editor, *18. Annual Symposium on Computer Applications in Medical Care*, pages 954–958. Hanley and Belfus, 1994.

[7] C. Combi, F. Pinciroli, and G. Pozzi. Object-Orientated DBMS Techniques for Time-Orientated Medical Record. *Medical Informatics*, 17(4):231–241, 1992.

[8] C. Combi, F. Pinciroli, and G. Pozzi. Temporal Clinical Data Modeling and Implementation for PTCA Patients in a OODBMS Environment. In *Proceedings Computers in Cardiology*, pages 505–508. IEEE Computer Press, 1994.

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

[10] A.K. Das and M.A. Musen. A Temporal Query System for Protocol-Directed Decision Support. *Methods of Information in Medicine*, 33:358–370, 1994.

[11] C.J. Date. *A Guide to SQL Standard.* Addison Wesley, 1987.

[12] U. Dayal and G. Wuu. A Uniform Approach to Processing Temporal Queries. In *Proc. 18th Int'l Conf. on Very Large Data Bases*, pages 407–418, August 1992.

[13] I.A. Goralwalla, Y. Leontiev, M.T. Özsu, and D. Szafron. A Uniform Behavioral Temporal Object Model. Technical Report TR-95-13, University of Alberta, May 1995.

[14] I.A. Goralwalla, Y. Leontiev, M.T. Özsu, and D. Szafron. Modeling Time: Back to Basics. Technical Report TR-96-03, University of Alberta, February 1996.

[15] B. Irani. Implementation, Design, and Development of the TIGUKAT Object Model. Master's thesis, University of Alberta, Department of Computing Science, Edmonton, Alberta, 1993. Available as University of Alberta Technical Report TR93-10.

[16] L.M. Jolicoeur, A.J. Jones-Grizzle, and J.G. Boyer. Guidelines for performing a pharmacoeconomic analysis. *American Journal of Hospital Pharmacy*, 49:1741–1747, July 1992.

[17] W. Kafer and H. Schoning. Realizing a Temporal Complex-Object Data Model. In *Proc. ACM SIGMOD Int'l. Conf. on Management of Data*, pages 266–275, 1992.

[18] M.G. Kahn, L.M. Fagan, and S. Tu. Extensions to the Time-Oriented Database Model to Support Temporal Reasoning in Medical Expert Systems. *Methods of Information in Medicine*, 30:4–14, 1991.

[19] M.G. Kahn, L.M. Fagan, and S. Tu. TQuery: A Context-Sensitive Temporal Query Language. *Computers and Biomedical Research*, 24:401–419, 1991.

[20] S.N. Khoshafian and G.P. Copeland. Object Identity. In *Proc. of the Int'l Conf on Object-Oriented Programming: Systems, Languages, and Applications*, pages 406–416, September 1986.

[21] M.T. Özsu, R.J. Peters, D. Szafron, B. Irani, A. Lipka, and A. Munoz. TIGUKAT: A Uniform Behavioral Objectbase Management System. *The VLDB Journal*, 4:100–147, August 1995.

[22] R.J. Peters. *TIGUKAT: A Uniform Behavioral Objectbase Management System.* PhD thesis, University of Alberta, 1994.

[23] R.J. Peters, A. Lipka, M.T. Özsu, and D. Szafron. An Extensible Query Model and Its Languages for a Uniform Behavioral Object Management System. In *Proc. Second Int'l. Conf. on Information and Knowledge Management*, November 1993.

[24] E. Rose and A. Segev. TOODM - A Temporal Object-Oriented Data Model with Temporal Constraints. In *Proc. 10th Int'l Conf. on the Entity Relationship Approach*, pages 205–229, October 1991.

[25] E. Rose and A. Segev. TOOSQL - A Temporal Object-Oriented Query Language. In *Proc. 12th Int'l Conf. on the Entity Relationship Approach*, pages 128–138, December 1993.

[26] R. Snodgrass. The Temporal Query Language, TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[27] R. Snodgrass and I. Ahn. A Taxonomy of Time in Databases. In *Proc. ACM SIGMOD Int'l. Conf. on Management of Data*, pages 236–246, May 1985.

[28] S.Y.W. Su and H.M. Chen. Modeling and Management of Temporal Data in Object-Oriented Knowledge Bases. In *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, pages HH1–HH18, Arlington, Texas, June 1993.

[29] G. Wuu and U. Dayal. A Uniform Model for Temporal Object-Oriented Databases. In *Proc. 8th Int'l. Conf. on Data Engineering*, pages 584–593, February 1992.

Appendix: Behavior Signatures for the Primitive Time Types

Type	Signatures	
T_interval	*B_lbOpen*:	T_boolean
	B_ubOpen:	T_boolean
	B_lb:	T_instant
	B_ub:	T_instant
	B_length:	T_span
	B_timeline:	T_timeline
	B_prev:	T_collection⟨T_interval⟩
	B_next:	T_collection⟨T_interval⟩
	B_precedes:	T_interval → T_boolean
	B_follows:	T_interval → T_boolean
	B_within:	T_interval → T_boolean
	B_meets:	T_interval → T_boolean
	B_overlaps:	T_interval → T_boolean
	B_starts:	T_interval → T_boolean
	B_finishes:	T_interval → T_boolean
	B_union:	T_interval → T_interval
	B_intersection:	T_interval → T_interval
	B_difference:	T_interval → T_interval
	B_subtract:	T_span → T_interval
	B_add:	T_span → T_interval
	B_expand:	T_span → T_interval
	B_shrink:	T_span → T_interval
T_instant	*B_lessthaneqto*:	T_instant → T_boolean
	B_greaterthaneqto:	T_instant → T_boolean
	B_elapsed:	T_instant → T_span
	B_subtract:	T_span → T_instant
	B_add:	T_span → T_instant
	B_intersection:	T_interval → T_instant
	B_difference:	T_interval → T_instant
	B_shrink:	T_span → T_instant
T_span	*B_lessthan*:	T_span → T_boolean
	B_greaterthan:	T_span → T_boolean
	B_lessthaneqto:	T_span → T_boolean
	B_greaterthaneqto:	T_span → T_boolean
	B_add:	T_span → T_span
	B_subtract:	T_span → T_span

Table 2: Behaviors on time intervals, time instants, and time spans.