University of Alberta

# A Uniform Behavioral Temporal Object Model

by

Iqbal A. Goralwalla, Yuri Leontiev, M. Tamer Özsu and Duane Szafron

Technical Report TR 95–13

July 1995

**DEPARTMENT OF COMPUTING SCIENCE**
**The University of Alberta**
**Edmonton, Alberta, Canada**

# A Uniform Behavioral Temporal Object Model[1]

Iqbal A. Goralwalla, Yuri Leontiev, M. Tamer Özsu and Duane Szafron

Laboratory for Database Systems Research

Department of Computing Science

University of Alberta

Edmonton, Alberta, Canada T6G 2H1

{iqbal,yuri,ozsu,duane}@cs.ualberta.ca

[1]This research has been supported by the Natural Sciences and Engineering Research Council of Canada under research grants OGP0951 and OGP8191. The second author's research is also supported by the Izaak Walton Killam Memorial Scholarship.

# Contents

## List of Figures

## List of Tables

## Abstract

In this work we present a uniform behavioral temporal object model which includes a rich and extensible set of types and behaviors to support various notions of time. Our temporal model supports the continuous and discrete domains of time. It also supports various specifications of time namely, time instants, time intervals, and time spans. Two issues which deal with temporal information that frequently arise in real-world applications are the need to consistently store and operate on temporal information that is comprised of different granularities, and the ability to represent indeterminate temporal information. Temporal indeterminacy arises when we know for certain that an event did occur, but exactly when it occured is unknown. Our model provides a uniform framework that supports different granularities of time and temporal indeterminacy. We show how different granularities can be converted to each other, and how this leads to temporal indeterminacy. We further show how continuous and discrete indeterminate time intervals, spans, and instants are consistently modeled in our framework.

**Keywords**: temporal databases, object models, granularity, incomplete information, temporal indeterminacy.

## 1 Introduction

Time is an inherent feature of applications that require the functionality of objectbase management systems (OBMSs)[1]. Most of the applications for which OBMSs are expected to provide support exhibit some form of temporality. Some examples are the following: in engineering databases there is a need to identify different versions of a design as it evolves; in multimedia systems the video images are timed and synchronized with audio; in office information systems documents are ordered based on their temporal relationships; in news agencies it is usually desirable to maintain histories of events pertaining to different situations which need news coverage. Thus, a temporal domain is a natural part of an OBMS. Temporal information usually involves:

- *Different granularities* – In most cases, information processed by an OBMS is available in multiple granularities. For example, a news agency that is providing coverage of an ongoing murder trial needs to represent the day-to-day history of the trial from the day it began up to the current date. It also needs to represent the history of the actual and alleged events preceding, during and following the murder. Some of these events are represented with the granularity of hours, while others have a granularity of minutes. For example, the accused alleges to be playing golf from $6pm\ June\ 12$ to $8pm\ June\ 12$, while the alleged time of murder is assumed to be $8:10pm\ June\ 12$. Another example would be a medical information system where the history of an admitted patient would be kept on a daily basis whereas the history of the condition of his body would be kept on an hourly basis.

- *Indeterminacy* – Often, it is known that an event did occur, but exactly when it occured is unknown. Consider the alleged time of murder in the trial that a news agency is covering. Both the prosecution and defense may be certain that the murder took place between $8:00pm\ June\ 12$ and $8:30pm\ June\ 12$, but are uncertain of the exact time. Additionally, it is known that the murder took between 2 and 5 minutes. However, the exact duration is unknown. These are examples of indeterminate temporal information. Temporal indeterminacy also arises from granularities. For example, if the alleged time of murder is $8:10pm\ June\ 12$ and we are interested in knowing at which second it occured, we can only conclude that it occured sometime between $8:10:00pm\ June\ 12$ and $8:10:59pm\ June\ 12$. Similarly, if it was noticed that the patient was dead at a particular hour, say $6am\ May\ 30$, one can only

[1]We prefer the terms "objectbase" and "objectbase management system" over the more popular terms "object-oriented database" and "object-oriented database management system", since not only data in the traditional sense are managed, but objects in general, which includes things such as code in addition to data.

reasonably conclude that he died sometime between $5:00am\ May$ 30 and $5:59am\ May$ 30.

There have been many object-oriented temporal model proposals (see, for example, [KC86, RS91, RS93, KS92, DW92, WD92, SC93, CG93]). These models differ in the functionality that they offer, but none of them provides support for uniformly handling different domains of time, granularities and indeterminate temporal information.

In this paper we describe a temporal object model which is sufficiently powerful to meet the requirements of applications that need an OBMS. Our work is conducted within the context of the TIGUKAT[2] system [ÖPS+95] that is currently under development at the University of Alberta. We exploit the behaviorality and uniformity of the TIGUKAT object model in incorporating time uniformly. The identifying characteristics of our temporal object model are the following:

1. Different time primitives are supported for specifying time. These are *instants*, *intervals*, and *spans*. The model supports four types of intervals viz [], [), (], and (). Various operations on time instants, time intervals, and time spans have been identified and modeled by a rich set of corresponding behaviors.

2. Both *discrete* and *continuous* domains of time are supported. This is in contrast to previous work which deals with only a single domain of time which is usually discrete [Sno92]. The need to support different time domains in a general OBMS is the emerging consensus among the temporal database research community [DSS94].

3. We consistently represent discrete time information with different granularities and continuous time information in the same framework.

4. We clearly define the conversion procedures needed to operate on time specifications with *mixed* granularities.

5. We uniformly represent and seamlessly operate on indeterminate and determinate temporal information by using a set-theoretic algebra to operate on all anchored time specifications.

These characteristics enable the design of a wide range of applications requiring different models of time to be carried out with ease and in a uniform manner.

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

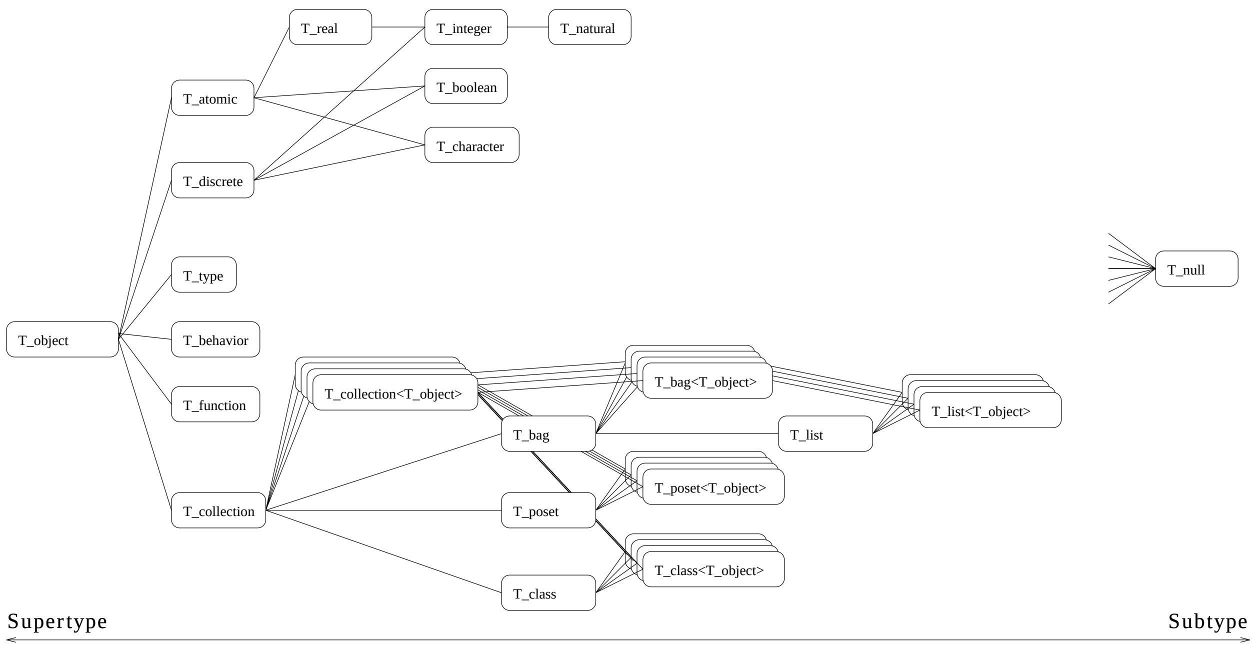

Figure 1: Primitive type system of the TIGUKAT object model.

The rest of the paper is organized as follows. In Section 2, we describe the TIGUKAT object model. In Section 3 we describe our temporal model and show how it supports different time domains, time specifications, granularities, and incomplete information. Section 4 shows how our temporal model is uniformly incorporated into TIGUKAT. In Section 5 we justify the richness of our temporal model by comparing it with other temporal object models. Section 6 concludes the paper and outlines future research.

## 2 The TIGUKAT Object Model

The TIGUKAT object model [Pet94] 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 type system of the TIGUKAT object model is shown in Figure 1.

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 is 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.

In addition to classes, a *collection* is defined as a general grouping construct. A *collection* is similar to a *class* in that it groups objects, but it differs in the following respects. First, object creation cannot occur through a collection; object creation occurs 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. Finally, the elements of a class are homogeneous up to inclusion polymorphism, while a collection may be heterogeneous in the sense that it may contain objects of types that are not in a subtype relationship with one another.

The type for classes (T_class) is a subtype of the type for collections (T_collection), and thus classes are specialized collections. This introduces a clean semantics between the two and allows the model to utilize both constructs in an effective manner. For example, queries can be performed over collections as well as classes.

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

## 3 The Temporal Object Model

Applications built on top of TIGUKAT require an extensible type system and may need different type semantics to model temporality. For example, TIGUKAT will be used in a news-on-demand multimedia information system that requires temporal support to represent synchronization requirements among video and audio [ÖSEV95]. Consequently, we provide a temporal model which includes a rich and extensible set of types and behaviors to support various notions of time. In this section we describe the various features of our temporal object model.

### 3.1 Temporal Domains

In our model, we support two different domains of time:

- *Discrete* domains map time to the set of integers. For any member of a discrete time domain, there is a unique successor and predecessor. For example, in a murder trial, the times associated with the trial history and the history of the actual and alleged events immediately preceding, during and following the alleged murder are discrete. In the trial history, the events which take place on each day of the trial are recorded in the objectbase. Hence, on any particular day in the history of the trial, we can look up what happened on the previous or following day.

- *Continuous* domains map time to the set of real numbers. Between any two members of a continuous time domain, another member exists. For example, reporters working with a news agency that is covering a murder trial enter information with respect to a global (virtual) clock (GST), which is continuous by definition. Hence, the times at which information is entered and updated in the objectbase by the reporters is from the continuous time domain.

Supporting both discrete and continuous domains of time is unique since the previous models deal with only a single domain of time, which is usually discrete. However, the continuous time domain is equally necessary and its importance in applications such as the one described above and real-time applications involving mathematics and physics cannot be ignored. Moreover, continuous time is essential in most of the current multimedia applications where the video and audio have to be synchronized for coherent presentation. The need for the continuous time domain is also emphasized in [Cho94] where it is argued that the continuous time domain can be used to substantially increase the efficiency of the representation of time as well as that of temporal query evaluation.

## 3.2 Temporal ontology

In the real world there are many cases when we have complete knowledge of the time or the duration of a particular event. For example, we know for certain the time of sunrise and sunset each day in a particular city. However, there are cases when the knowledge of the time or the duration of a particular event is known only to a certain extent. For example, we do not know the exact moment when the Earth was formed but we do know the time frame for this event. Since the ultimate purpose of a temporal model is to represent temporal information about real events, it is desirable for such a model to be able to capture different kinds of complete and incomplete information.

In our model, temporal information can be unanchored (represented by determinate and indeterminate spans) or anchored (represented by determinate and indeterminate instants, intervals and time sets based on them). These two categories of temporal information are discussed in the following sections.

### 3.2.1 Unanchored Temporal Information

We let $T_u$ be the universal set of all possible unanchored specifications of time; 5 hours, 10 days, 2 to 3 months, etc., are all unanchored specifications of time. 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, with a number of operations defined on it:

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

#### Determinate Spans

A *determinate span* represents complete information about a duration of time. The maximum time allowed for students to complete their *Introduction to Database Management Systems* examination, for example, is a determinate span.

#### Indeterminate Spans

An *indeterminate span* represents incomplete information about a duration of time. It has lower and upper bounds that are determinate spans. $1\ day \sim 2\ days$, for example, is an indeterminate span that can be interpreted as "a time period between one and two days." Any determinate span can be represented as a special kind of indeterminate span with identical lower and upper bounds. We do not allow disjoint indeterminate spans, e.g., "this event lasted for either 2 or 3 days but not 2 days 6 hours." An indeterminate span can be either continuous or discrete. In the latter case both lower and upper boundaries must be discrete spans (not necessarily of the same granularity). Since discrete spans are a special case of continuous ones, indeterminate spans with one discrete and one continuous boundary are allowed and are treated as continuous ones.

## Granularities

In many applications, it is desirable to operate on times that have different granularities (days, months, years, etc.). One way to meet this requirement is to provide system support for a large number of granularities. However, this has high overhead, and inevitably, there will be applications that will need granularities beyond a reasonable set provided by the system. It is, therefore, important for the model to be extensible and not limited to a predefined set of granularities that can be used to represent time.

Generally speaking, a *granularity* is a unit of measurement for the time durations. Granularities are also commonly used to express a lack of information about the particular time of an event. For example, the statement "I am flying to Calgary on January 5th, 1995," implies that the flight will take place *some time* on January 5th. In this case the granularity that is used to represent the time of the flight (1 day) is also a period of indeterminacy. Note that no indeterminacy arises when the duration of time rather than a particular moment is concerned. In other words, 1 day as a duration of time carries no indeterminacy.

In our model, discrete granularity is a special kind of determinate time span that can be used as a unit of time. For example, the granularity of days behaves similarly to the time span 1 *day*. We will use the letter $G$ to refer to a particular granularity (e.g. $G_{day}$ is a granularity of days). Every operation that can be performed on a span can also be performed on a granularity. For example, to obtain a time span of 5 days we would multiply the granularity of days by the integer 5: $5 \cdot G_{day}$. To obtain a time span of 2 *months and* 5 *days*, we would add the span of 2 *months* to the span of 5 *days*: $2 \cdot G_{month} + 5 \cdot G_{day}$. In general, we define a discrete determinate span as a finite sum:

$$S_{discr} = \sum_{i=1}^{N} (K_i \cdot G_i) \tag{1}$$

where $K_i$ are integer coefficients and $G_i$ are distinct granularities.

Since a granularity is a special kind of a time span it is meaningful to compare two granularities with each other. If one granularity is greater than the other (as a time span) we say that the former is the *coarser* and the latter is the *finer* between the two. We assume that we can always compare two granularities with each other, thus the set of all possible granularities is totally ordered (with respect to the comparison described above). For example, the span of 1 *day* is shorter ($<$) than the span of 1 *month* and therefore the granularity of days is finer than the granularity of months.

In a temporal model where times with different granularities are supported we need to be able to convert one granularity to another. Is it always possible to convert a time span from the coarser to the finer granularity without loss of information? The answer, perhaps surprisingly, is no. To illustrate this point let us consider the following conversions. The conversion of the time span 1 *hour* to the granularity of minutes is exact and will result in the time span of 60 *minutes*. However, the conversion of the time span 1 *month* to the finer granularity of days can not possibly be an exact one. Should the resulting time span be 31, 30, 29 or 28 days? We cannot tell unless we know exactly which month is involved. Since a time span is *unanchored* this information is not available. Of course, we could convert 1 *month* to the indeterminate span 28 *days* $\sim$ 31 *days* but in this case the conversion is not exact and some information is lost. Therefore, the set of all granularities is not totally ordered with respect to the binary relation "exactly convertible to." In fact, this relation specifies a partial order on the granularities which is a suborder of the comparison-based order described earlier. Indeterminate spans are considered in more detail in the next section.

In order to be able to carry out the conversion (whether exact or inexact) of a time span of granularity $G_A$ to a granularity $G_B$ we define two functions called *lower bound factor* ($lbf(G_A, G_B)$) and *upper bound factor* ($ubf(G_A, G_B)$). The lower (upper) bound factor of $G_A$ and $G_B$ is the minimum (maximum) number of $G_B$ units that can form 1 $G_A$ unit. For example, $lbf(G_{month}, G_{day}) = 28$ and $ubf(G_{month}, G_{day}) = 31$. Both factors coincide in the case of exact conversion. For instance, $lbf(G_{hour}, G_{minute}) = ubf(G_{hour}, G_{minute}) = 60$. Note that it is not a requirement that $G_B$ be finer than $G_A$. For example, $lbf(G_{day}, G_{month}) = 1/31$ and $ubf(G_{day}, G_{month}) = 1/28$. The user can define new granularities in terms of existing ones. For example, the new granularity *decade* could be defined in terms of the existing granularity year using $lbf(G_{decade}, G_{year}) = ubf(G_{decade}, G_{year}) = 10$. In general, for any $G_A$, $G_B$ and $G_C$ the following hold:

1. $lbf(G_A, G_A) = ubf(G_A, G_A) = 1$
2. $0 < lbf(G_A, G_B) \leq ubf(G_A, G_B)$

3. $lbf(G_A, G_B) \cdot ubf(G_B, G_A) = 1$

4. $lbf(G_A, G_B) \cdot lbf(G_B, G_C) \leq lbf(G_A, G_C)$

5. $ubf(G_A, G_B) \cdot ubf(G_B, G_C) \geq ubf(G_A, G_C)$

(The last rule is a consequence of the first four)

We now define a conversion of a discrete determinate span to any given granularity $G_A$. The conversion of a span of the form depicted in formula (1) to a granularity $G_A$ results in an indeterminate span with lower bound

$$\lfloor \sum_{i=1}^{N} L_i \rfloor \cdot G_A \tag{2}$$

and upper bound

$$\lceil \sum_{i=1}^{N} U_i \rceil \cdot G_A \tag{3}$$

where

$$L_i = K_i \cdot lbf(G_i, G_A) \tag{4}$$

and

$$U_i = K_i \cdot ubf(G_i, G_A) \tag{5}$$

To illustrate the conversion described above, let us convert the time span 2 *months and* 45 *hours* to the granularity of days ($G_{days}$). First we represent the given span in the form given in formula (1):

$$2 \cdot G_{months} + 45 \cdot G_{hours}$$

In this span, $K_1 = 2, K_2 = 45, G_1 = G_{months}, G_2 = G_{hours}$. We now use the formulas (4) and (5) to compute $L_1, L_2, U_1, U_2$:

$$\begin{aligned}
L_1 &= K_1 \cdot lbf(G_1, G_{days}) \\
&= 2 \cdot lbf(G_{months}, G_{days}) \\
&= 2 \cdot 28 \\
&= 56 \\
U_1 &= K_1 \cdot ubf(G_1, G_{days}) \\
&= 62 \\
L_2 &= K_2 \cdot lbf(G_2, G_{days})
\end{aligned}$$

$$
\begin{aligned}
&= 45 \cdot lbf(G_{hours}, G_{days}) \\
&= 45/ubf(G_{days}, G_{hours}) \\
&= 45/24) \\
&= 1.875 \\
U_2 &= K_2 \cdot ubf(G_2, G_{days}) \\
&= 1.875
\end{aligned}
$$

Now we compute the lower and upper boundary of the resulting indeterminate span according to formulas (2) and (3), respectively:

$$
\begin{aligned}
\mathit{lower\ bound} &= \lfloor L_1 + L_2 \rfloor \cdot G_{days} \\
&= \lfloor 56 + 1.875 \rfloor \cdot G_{days} \\
&= 57 \cdot G_{days} \\
\mathit{upper\ bound} &= \lceil U_1 + U_2 \rceil \cdot G_{days} \\
&= \lceil 62 + 1.875 \rceil \cdot G_{days} \\
&= 64 \cdot G_{days}
\end{aligned}
$$

Hence, the result of our conversion is the indeterminate discrete time span 57 *days* $\sim$ 64 *days.*

So far we have discussed discrete spans only. However, our temporal model allows the co-existence of discrete and continuous time spans (as well as time instants, etc.). Therefore, it is relevant to define conversions between discrete and continuous spans as well. Continuous determinate spans are of the form

$$S_{cont} = \sum_{i=1}^{N} (R_i \cdot G_i) \qquad (6)$$

where $R_i$ are real coefficients and $G_i$ are distinct granularities. This formula is a generalization of formula (1) for the case of real coefficients. This means that it is always possible to convert any discrete span to a continuous span *exactly.* Therefore, no conversion from discrete to continuous spans is required. To convert a continuous span to a discrete granularity, we use the same formulas (2–5) as for the discrete case. For example, the conversion of the continuous time span 0.31 *hours* to the granularity of minutes results in the discrete indeterminate time span 18 *minutes* $\sim$ 19 *minutes.*

In this section we have defined our notion of granularities. We have also given the rules for conversion of time spans between different granularities. We saw how this coversion led to incomplete temporal information in the form of an indeterminate time span. In the next section we discuss incomplete temporal information in detail and show how it is supported within our framework.

### 3.2.2 Anchored Temporal Information

#### Overview

We model time by letting $T_a$ be the universal set of all possible anchored specifications of time. That is, July 31, [1967,1968), t26, [t7,t17], 9:17:54:20, etc., are all anchored specifications of time. We identify a *time interval* as the basic anchored specification of time. A time interval is a duration of time between two specific anchor points which stand for the lower and upper bounds of the interval (e.g., [June 15,July 31], [t5,t9), etc.). A wide range of operations can be performed on time intervals:

1. Unary operators can be defined on a time interval which return the lower bound, upper bound and length of the time interval.
2. A rich set of ordering operations between intervals can be defined [All84]. These are depicted in Figure 2 and are self-explanatory.
3. 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.
5. A time interval can expand or shrink by a specified time duration.

A *time instant* (*moment*, *chronon*, etc.) is a specific anchored moment in time (e.g., 67, $t_7$, June 15). A time instant is a special case of a (closed) time interval which has the same lower and upper bounds. For example, June 15 = [June 15,June15]. A wide range of operations can be performed on time instants:

1. A time instant can be compared with another time instant with the transitive comparison operators $<$ and $>$.
2. A time instant can be subtracted from another time instant to find the time duration between the two.

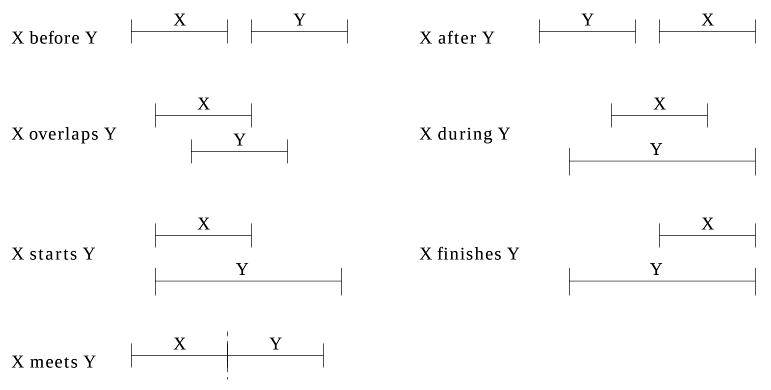

Figure 2: Different types of ordering relations between intervals.

3. A time duration can be added or subtracted from a time instant to return another time instant.

4. A time instant can be compared with a time interval to check if it falls before, within or after the time interval.

**Set-theoretic framework**

In this section, we provide a set-theoretic framework for modeling anchored temporal information. We then, in subsequent sections, define determinate and indeterminate time instants and intervals and show how they are mapped to the set-theoretic framework.

Our model of anchored temporal information is based on set theory. The set-theoretic foundations behind our model are the following. We define a *time set* as an ordered pair of two sets: $S_d$ (definite time set) and $S_p$ (probable time set) related by the condition $S_d \subseteq S_p$. We denote the time set as $F = \langle S_d, S_p \rangle$. This construct has the following semantics: the event definitely took place during $S_d$ and probably took place during $S_p$. For example, if we know that a certain building was on fire from $t_2$ to $t_3$ and we also know that it was *not* on fire before $t_1$ or after $t_4$, then the time set corresponding to this information about the time of the fire will be $F = \langle \{t \mid t_2 \leq t \leq t_3\}, \{t \mid t_1 \leq t \leq t_4\} \rangle$. In this paper, we do not consider probability distributions. We define the usual set-theoretic operations (complement, union and intersection) as follows: $\bar{F} = \langle \bar{S}_p, \bar{S}_d \rangle$, $F^1 \cup F^2 = \langle S_d^1 \cup S_d^2, S_p^2 \cup S_p^2 \rangle$, and $F^1 \cap F^2 = \langle S_d^1 \cap S_d^2, S_p^2 \cap S_p^2 \rangle$. Other set-theoretic operations (difference, subset, etc.) can be defined in terms of these three.

In our model, we use only time sets that can be represented as a union of any finite number of determinate and indeterminate instants and intervals (these will be defined shortly). It is easy to see that the set of all time sets in our model is closed with respect to set-theoretic operations (complement, union and intersection) and their combinations.

## Continuous time instants and intervals

Continuous instants are just points on the (continuous) line of all anchored time specifications. They are totally ordered by the relation "later than." The time set corresponding to a continuous instant $t_{cont}$ is $\langle\{t_{cont}\},\{t_{cont}\}\rangle$ ($S_d = S_p = \{t_{cont}\}$). It is possible to add (subtract) a determinate time span to (from) a continuous time instant. This operation yields another continuous time instant and has the meaning of displacing a given instant along the line of anchored times, by the time span[4].

In our model, there are two kinds of continuous intervals: *determinate intervals* and *indeterminate intervals*. In addition, four categories of intervals ($(), [), (]$, and $[]$) are supported for each kind. Every interval has lower and upper boundaries which are continuous instants. We use the usual interval notation for indeterminate intervals with a tilde instead of a comma. For example, a closed indeterminate time interval with the lower boundary $t_1$ and upper boundary $t_2$ is denoted as $[t_1 \sim t_2]$. The closed determinate interval with the same boundaries is denoted as $[t_1, t_2]$. The difference between determinate and indeterminate time intervals is that a determinate interval denotes an event that occured during each instant of the interval whereas an indeterminate interval denotes an event that *could* occur at each instant of the interval. Therefore determinate and indeterminate intervals have different mappings to the time sets. A determinate interval $I_{det}$ is mapped to the time set $\langle I_{det}, I_{det}\rangle$ ($S_d = S_p$). An indeterminate interval $I_{indet}$ is mapped to the time set $\langle\emptyset, I_{indet}\rangle$ ($S_d = \emptyset$).

## Discrete time instants and intervals

There are two possible interpretations of a discrete time instant. We will illustrate these by using the following example. Assume that somebody has been on a train the whole day of January 5th, 1987. To express this fact we will use the determinate time instant 5 *January* $1987_{det}$ (which means *the whole day of*). However, if somebody is leaving for Paris on January 5th, 1987 we would represent this as an indeterminate time instant 5 *January* $1987_{indet}$ (which means *some time on that day*). Hence, in our model there are two different kinds of discrete instants: *determinate*

[4]Note that this operation can be applied to any anchored time specification and has exactly the same meaning for all of them.

and *indeterminate* that correspond to the two different interpretations, respectively. Essentially, a determinate (indeterminate) discrete time instant behaves like a determinate (indeterminate) continuous interval. For example, the time instant $5\ January\ 1987_{det}$ mentioned above is analogous to the interval $[5\ January\ 1987_{cont}, 6\ January\ 1987_{cont})$.

Every discrete instant has a granularity associated with it. This granularity determines the mapping of the given discrete time instant to the continuous time domain. We map the discrete determinate (indeterminate) time instant $t_{discr}$ with a granularity $G_t$ to the continuous determinate (indeterminate) interval $[t_{cont}, t_{cont} + G_t)$ ($[t_{cont}\ \sim\ t_{cont} + G_t)$). Here $t_{cont}$ denotes the counterpart of $t_{discr}$ in the continuous domain. This is exemplified by the mapping of the discrete determinate instant $5\ January\ 1987_{det}$ to the interval $[5\ January\ 1987, 6\ January\ 1987)$. In this case $G_t = G_{days} = 1\ day$. The upper bound of the resulting interval is defined to be open to ensure that different time instants with the same granularity do not overlap. The resulting determinate (indeterminate) continuous time intervals can then be mapped to the set-theoretic framework as shown in the previous section.

Discrete time instants can be used to form *discrete time intervals*. Since we have determinate and indeterminate discrete instants, we also have determinate and indeterminate discrete intervals. Determinate (indeterminate) time instants can be used as boundaries of determinate (indeterminate) time intervals. To map discrete determinate intervals to the continuous ones, we use the following algorithm:

1. Convert the two boundaries of the discrete interval (which are discrete instants) into continuous intervals. Let us assume the lower bound of the discrete interval is $lb_{discr}$ and has a granularity $G_{lb}$. Similarly, let us assume the upper bound of the discrete interval is $ub_{discr}$ and has a granularity $G_{ub}$. Then:
   (a) $lb_{discr} \Rightarrow [lb_{cont}, lb_{cont} + G_{lb})$
   (b) $ub_{discr} \Rightarrow [ub_{cont}, ub_{cont} + G_{ub})$
2. Rewrite the resulting structure (which is a kind of interval whose boundaries are also intervals) to a continuous interval using the following rules:
   (a) $[[a, b), [c, d)] \Rightarrow [a, d)$
   (b) $[[a, b), [c, d)) \Rightarrow [a, c)$
   (c) $([a, b), [c, d)] \Rightarrow [b, d)$

(d) $([a,b),[c,d)) \Rightarrow [b,c)$

To illustrate this algorithm we will consider the mapping of the following intervals, ($January$ 1994, 15 $March$ 1994] and [$February$ 1994, 16 $March$ 1994) to their continuous counterparts.

$$(January\ 1994,\ 15\ March\ 1994]$$
$$\Rightarrow\ (\ [January\ 1994_{cont}, February\ 1994_{cont}),\ [15\ March\ 1994_{cont}, 16\ March\ 1994_{cont})\ ]$$
$$\Rightarrow\ [February\ 1994_{cont}, 16\ March\ 1994_{cont})$$

According to Step 1 of the algorithm, we first convert $January$ $1994_{det}$ to the continuous interval [$January$ $1994_{cont}$, $February$ $1994_{cont}$) and 15 $March$ $1994_{det}$ to the continuous interval [15 $March$ $1994_{cont}$, 16 $March$ $1994_{cont}$). Next, we use rule (c) of Step 2 to rewrite

$$(\ [January\ 1994_{cont}, February\ 1994_{cont}),\ [15\ March\ 1994_{cont}, 16\ March\ 1994_{cont})\ ]$$

as [$February$ $1994_{cont}$, 16 $March$ $1994_{cont}$).

$$[February\ 1994,\ 16\ March\ 1994)$$
$$\Rightarrow\ [\ [February\ 1994_{cont}, March\ 1994_{cont}),\ [16\ March\ 1994_{cont}, 17\ March\ 1994_{cont})\ )$$
$$\Rightarrow\ [February\ 1994_{cont}, 16\ March\ 1994_{cont})$$

Here we use the same technique as before. The only difference is the usage of rule (b) instead of rule (c) in Step 2. This example shows that in our framework the (discrete) intervals ($January$ 1994, 15 $March$ 1994] and [$February$ 1994, 16 $March$ 1994) are equivalent. Intuitively, this is what is expected.

The mapping of discrete indeterminate intervals to continuous ones is similar to the one defined above but with tilde replacing the comma. For example, let us convert the indeterminate interval [$January$ 1993 $\sim$ 1994] to the continuous domain.

$$[January\ 1993\ \sim\ 1994]$$
$$\Rightarrow\ [\ [January\ 1993_{cont}\ \sim\ February\ 1993_{cont})\ \sim\ [1994_{cont}\ \sim\ 1995_{cont})\ ]$$
$$\Rightarrow\ [January\ 1993_{cont}\ \sim\ 1995_{cont})$$

Here we first convert $January$ $1993_{indet}$ to [$January$ $1993_{cont}$ $\sim$ $February$ $1993_{cont}$) and $1994_{indet}$ to [$1994_{cont}$ $\sim$ $1995_{cont}$). Then we use the analogue of the rule (a) of Step 2 for the indeterminate case to rewrite the result in an interval form.

Having defined indeterminate instants and intervals we will briefly come back to the problem of conversions between different granularities. The problems that we have encountered while dealing with spans do not arise here since instants as well as intervals are *anchored*. For example, if we want to express January A.C. in days we can do so since we know exactly which years (leap or not) and which months we are talking about. However, if we want to express $January\ 0001\ A.C._{indet}$ in days we will encounter another problem, namely that of indeterminacy. The indeterminacy implicit in an indeterminate time instant of the granularity of months is one month and that is different from the indeterminacy implicit in an indeterminate time instant with a granularity of days, which is one day. Therefore in our example we will obtain not an indeterminate time instant, but an indeterminate time interval with both boundaries being discrete indeterminate time instants with a granularity of days: $[1st\ January\ 0001\ A.C._{indet} \sim 31st\ January\ 0001\ A.C._{indet}]$. It is easy to see that if we want to convert from finer to coarser granularities we will have no problems at all. For example, the conversion of $15\ January\ 0001\ A.C._{indet}$ to months would yield $January\ 0001\ A.C._{indet}$. Here we have lost some information as we made a conversion to coarser granularity. The actual conversion process of an instant (interval) from one granularity to another necessitates us to take a more closer look at how we represent calenders in our model. We leave this to a forthcoming work [GLÖS95] in which we will talk about calenders in detail and also show how operations (add and subtract) between instants (intervals) and spans of different granularities are carried out.

## 4 Incorporating the Temporal Model in TIGUKAT

In this section we describe how the temporal model introduced in Section 3 is incorporated into the TIGUKAT object model. Two features of the model that are important for this discussion are its uniformity and its behavioral nature. The TIGUKAT object model is uniform in the sense that all entities (both system and user defined) are modeled as first-class objects. This uniformity extends to the schema objects (types, classes, etc) as well making the system reflective [PÖ93]. The behavioral nature of the model defines the execution model – all access to objects is by means of the application of behaviors (which are themselves first-class objects) to objects. The implementation of behaviors is by means of (stored or computed) functions. This eliminates the need to differentiate between attributes and methods and provides for both behavioral and implementation inheritance.

Consistent with TIGUKAT philosophy, every time specification that has been introduced in this paper has a corresponding TIGUKAT type. We denote these types by the prefix `T_` (for example,

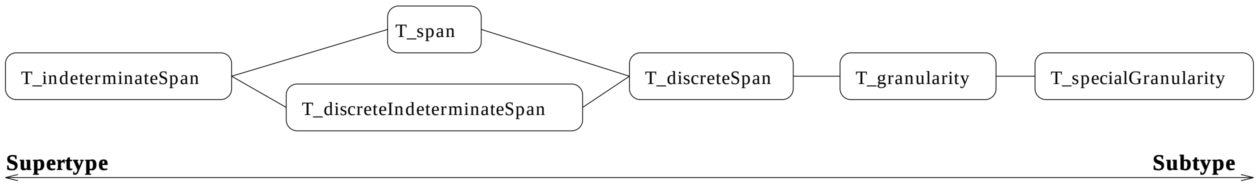

Figure 3: Span and granularity types.

| | | |
|---|---|---|
| `T_indeterminateSpan` | *B_lessthan*: | `T_indeterminateSpan` → `T_boolean` |
| | *B_greaterthan*: | `T_indeterminateSpan` → `T_boolean` |
| | *B_add*: | `T_span` → `T_indeterminateSpan` |
| | *B_subtract*: | `T_span` → `T_indeterminateSpan` |
| | *B_lowerBound*: | `T_span` |
| | *B_upperBound*: | `T_span` |
| `T_discreteIndeterminateSpan` | *B_add*: | `T_discreteSpan` → `T_discreteIndeterminateSpan` |
| | *B_subtract*: | `T_discreteSpan` → `T_discreteIndeterminateSpan` |
| | *B_lowerBound*: | `T_discreteSpan` |
| | *B_upperBound*: | `T_discreteSpan` |
| `T_span` | *B_add*: | `T_span` → `T_span` |
| | *B_subtract*: | `T_span` → `T_span` |
| | *B_units*: | T_collection⟨T_granularity⟩ |
| | *B_coefficient*: | `T_granularity` → `T_real` |
| | *B_multiply*: | `T_real` → `T_span` |
| | *B_divide*: | `T_real` → `T_span` |
| | *B_convertTo*: | `T_granularity` → `T_discreteIndeterminateSpan` |
| `T_discreteSpan` | *B_add*: | `T_discreteSpan` → `T_discreteSpan` |
| | *B_subtract*: | `T_discreteSpan` → `T_discreteSpan` |
| | *B_coefficient*: | `T_granularity` → `T_integer` |
| | *B_multiply*: | `T_integer` → `T_discreteSpan` |
| | *B_succ*: | `T_discreteSpan` |
| | *B_pred*: | `T_discreteSpan` |

Table 1: Behaviors defined on time spans.

`T_interval` is the type of all continuous intervals). Types relevant to the representation of temporal information are depicted in Figures 3-8 along with their subtyping relationships. Likewise, every operation defined in this paper has a corresponding TIGUKAT behavior. These behaviors are discussed in the following sections and are listed along with their signatures in Tables 1–7. We now show the actual mapping between various temporal notions introduced so far and TIGUKAT types in our temporal model.

## 4.1 Time Spans

We start with the types related to the notion of a *time span*. These are shown in Figure 3. The various behaviors on time spans together with their signatures are shown in Table 1.

The type `T_indeterminateSpan` is introduced to model continuous indeterminate time spans. Behaviors defined on `T_indeterminateSpan` include *B_lessthan* and *B_greaterthan* which model the comparison operations on time spans. Behaviors *B_add*, *B_subtract* add and subtract allow continuous determinate spans to be added to and subtracted from continuous indeterminate spans, respectively. For example:

1. $(5\ days \sim 7\ days) \cdot B\_add(1\ day) \rightarrow 6\ days \sim 8\ days$
2. $(5\ days \sim 1\ month) \cdot B\_add(2\ days) \rightarrow 7\ days \sim (1\ month + 2\ days)$
3. $(1\ month \sim 2\ months) \cdot B\_subtract(30\ days) \rightarrow (1\ month - 30\ days) \sim (2\ months - 30\ days)$
4. $(1\ month - 30\ days) \sim (2\ months - 30\ days) \cdot B\_subtract(1\ month - 30\ days + 5\ hours)$ $\rightarrow -5\ hours\ \sim (1\ month - 5\ hours)$
5. $(1\ month - 30\ days) \sim 15\ days \cdot B\_add(5\ hours) \rightarrow (1\ month - 30\ days + 5\ hours) \sim (15\ days + 5\ hours)$

Subtraction leads to the notion of negative spans. In our model, both positive and negative spans are allowed. This allows us to carry out the subtraction operation between spans of different granularities which could result in either a positive or negative span, for example, $1\ month.B\_subtract(30\ days)$. The unary behaviors, *B_lowerBound* and *B_upperBound* return the lower and upper boundaries (which are continuous determinate spans) of a continuous indeterminate time span, respectively. Note that in our model we require that the lower bound be less than or equal to the upper bound.

`T_indeterminateSpan` has two direct subtypes: `T_discreteIndeterminateSpan` and `T_span`. The latter corresponds to the notion of a *continuous determinate span*. This subtyping relationship has the following justification: Every continuous determinate span can be treated as an indeterminate one (with identical lower and upper bounds) and every discrete span can be treated as a continuous one.

In `T_discreteIndeterminateSpan`, the behaviors *B_add* and *B_subtract* take a discrete determinate span as an argument and return a discrete indeterminate span as the result. Furthermore, the unary behaviors *B_lowerBound* and *B_upperBound* are refined to return a discrete determinate span.

Behaviors *B_add* and *B_subtract* are refined in `T_span` to take a continuous determinate span as an argument and return a continuous determinate span as the result. Behaviors *B_units*, *B_coefficient*, *B_multiply*, *B_divide* and *B_convertTo* in `T_span` are used in the conversion process of a time span to a specific granularity as shown in Section 3.2.1. *B_units* returns a totally ordered collection of granularities in a time span. For example, the behavior application ($1\ month + 5\ days$)$\cdot$ *B_units* returns $\langle\ G_{day}, G_{month}\ \rangle$. The behavior *B_coefficient* returns the (real) coefficient of a time span given a specific granularity. For example, ($1\ month + 5\ days$)$\cdot$ *B_coefficient*($G_{day}$) returns back 5. Behaviors *B_multiply* and *B_divide* are basically used in the conversion process. The *B_convertTo* behavior is derived from the rest of the behaviors in `T_span` and essentially converts a determinate time span to an indeterminate time span with the specified granularity. For example, ($2\ months + 45\ hours$)$\cdot$ *B_convertTo*($G_{days}$) gives the indeterminate time span $57\ days\ \sim\ 64\ days$. The details of this conversion are shown in Section 3.2.1.

The type `T_discreteSpan` is defined as a subtype of the `T_indeterminateSpan`, `T_discreteIndeterminateSpan` and `T_span` types described above. Behaviors *B_add* and *B_subtract* are refined in `T_discreteSpan` to take a discrete determinate span as an argument and return a discrete determinate as a result. Behavior *B_coefficient* is refined to return the integer coefficient of a discrete time span and the *B_multiply* behavior is refined to multiply an integer by a discrete time span. Behaviors *B_succ* and *B_pred* are defined in `T_discreteSpan` to return the next or previous discrete time span of a particular discrete time span. For example, ($2\ months + 45\ hours$)$\cdot$ *B_succ* returns the time span $3\ months + 46\ hours$ while ($2\ months + 45\ hours$)$\cdot$ *B_pred* returns the time span $1\ month + 44\ hours$.

Recall that a *granularity* in our framework is a special kind of a determinate span. Therefore, we define the type `T_granularity` as a subtype of `T_discreteSpan`. Instances of `T_granularity` represent the different kinds of discrete granularities, e.g., *years*, *hours*, *months*. The user can create new granularities as instances of `T_granularity`. Since the set of all granularities is totally ordered with respect to the comparison operators ($<, >$), it would be expected of the user to specify which of the existing granularities it is immediately *finer* than. The system would then uniformly update the total order on the set of granularities. It is also the user's responsibility to provide the system with the lower (upper) bound factors of the new granularity with respect to other, already existing, granularities. These factors would then be used by the system in operations involving multiple granularities, which may require the new granularity to be converted to other granularities. For example, let us assume that we have $\{G_{year}, G_{century}\}$ as the existing set of granularities in our system. Suppose the user decides he needs a new granularity, $G_{decade}$ and specifies that $G_{decade} < G_{century}$, $G_{decade} > G_{year}$. The system would then update the set of granularities to reflect the addition of $G_{decade}$. The totally ordered set of granularities would now be $\{G_{year}, G_{decade}, G_{century}\}$. For conversion purposes, the user also specifies that $lbf(G_{decade}, G_{year}) = ubf(G_{decade}, G_{year}) = 10$

| | | |
|---|---|---|
| `T_granularity` | *B_leq*: | `T_granularity` → `T_boolean` |
| | *B_geq*: | `T_granularity` → `T_boolean` |
| | *B_lowerBound*: | `T_granularity` |
| | *B_upperBound*: | `T_granularity` |
| | *B_exactlyConvertibleTo*: | `T_granularity` → `T_boolean` |
| | *B_commonUnit*: | `T_granularity` → `T_granularity` |
| | *B_lbf*: | `T_granularity` → `T_real` |
| | *B_ubf*: | `T_granularity` → `T_real` |

Table 2: Behaviors defined on granularity.

and $lbf(G_{century}, G_{decade}) = ubf(G_{century}, G_{decade}) = 10$. Behaviors on granularities are shown in Table 2.

Behaviors *B_leq* and *B_geq* in `T_granularity` define an ordering between different granularities. The *B_lowerBound* and *B_upperBound* behaviors are refined accordingly to return a granularity as the lower and upper bound of a granularity. Since the granularity is determinate, these behaviors return the same value. The behavior *B_exactlyConvertibleTo* checks if a granularity is exactly convertible to another granularity. For example, $G_{day}$· *B_exactlyConvertibleTo*($G_{hour}$) returns `True`, while $G_{month}$· *B_exactlyConvertibleTo*($G_{day}$) returns `False`. Lastly, the *B_commonUnit* behavior returns the common granularity that two granularities can be converted to. For example, $G_{month}$· *B_commonUnit*($G_{day}$) returns $G_{day}$, while $G_{week}$· *B_commonUnit*($G_{month}$) also returns $G_{day}$. `T_granularity` also defines new behaviors, *B_ubf* and *B_lbf*. These return the upper and lower bound factors (see Section 3.2.1) of two discrete granularities, respectively. For example, $G_{month}$· *B_lbf*($G_{day}$) returns 28 while $G_{month}$· *B_ubf*($G_{day}$) returns 31. To represent the continuous granularity, we define a subtype of `T_granularity`, `T_specialGranularity`. There is only one continuous granularity. The continuous granularity is the granularity of time spans (instants) which have no indeterminacy, namely continuous spans (instants).

## 4.2 Time Sets

Since a time set is represented as a union of any finite number of determinate and indeterminate instants and intervals, we define the `T_timeSet` type to represent a time set and make it the supertype of all interval and instant types as we will show in the following sections. `T_timeSet` defines the comparison behaviors *B_precedes*, *B_follows*, *B_within* and *B_overlaps* which model the different types of ordering relations between time intervals shown in Figure 2. Behaviors *B_add* and *B_subtract* in `T_timeSet` add or subtract a time span from a time set, respectively and return another time set as a result. These behaviors essentially add (subtract) a time span to (from)

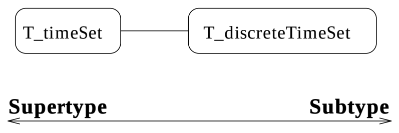

Figure 4: Time set types.

| | | |
|---|---|---|
| T_timeSet | *B_add:* | T_span → T_timeSet |
| | *B_subtract:* | T_span → T_timeSet |
| | *B_length:* | T_indeterminateSpan |
| | *B_complement:* | T_timeSet |
| | *B_within:* | T_timeSet → T_boolean |
| | *B_union:* | T_timeSet → T_timeSet |
| | *B_less:* | T_timeSet → T_boolean |
| | *B_greater:* | T_timeSet → T_boolean |
| | *B_overlaps:* | T_timeSet → T_boolean |
| | *B_intersection:* | T_timeSet → T_timeSet |
| | *B_difference:* | T_timeSet → T_timeSet |
| T_discreteTimeSet | *B_add:* | T_discreteSpan → T_discreteTimeSet |
| | *B_subtract:* | T_discreteSpan → T_discreteTimeSet |
| | *B_length:* | T_discteteIndeterminateSpan |
| | *B_complement:* | T_discreteTimeSet |
| | *B_union:* | T_discreteTimeSet → T_discreteTimeSet |
| | *B_intersection:* | T_discreteTimeSet → T_discreteTimeSet |
| | *B_difference:* | T_discreteTimeSet → T_discreteTimeSet |

Table 3: Behaviors defined on time set.

each of elements (instants and/or intervals) of a time set. Behaviors *B_add* and *B_subtract* are kind of displacement behaviors in that they cause a time set to shift by a certain duration of time. All set-theoretical operations on time sets discussed in Section 3.2.1 are defined as behaviors on `T_timeSet` (*B_complement*, *B_union*, *B_intersection*, etc.) and inherited by all interval and instant types which are described in the following sections. To model discrete time sets, we define the `T_discreteTimeSet` as a direct subtype of `T_timeSet` as shown in Figure 4.

`T_discreteTimeSet` inherits all its behaviors from `T_timeSet` and refines them to reflect discrete time sets. Behaviors on time sets are shown in Table 3.

## 4.3 Time Intervals

According to the theory described in Section 3.2.1, an interval is a special kind of a time set (a time set that is comprised of just one interval). We define the type `T_generalInterval` that defines behaviors common for all intervals as a subtype of `T_timeSet`. Behaviors defined on `T_generalInterval` are shown in Table 4.

|  |  |  |
|---|---|---|
| T_generalInterval | *B_add:* | T_span → T_generalInterval |
|  | *B_subtract:* | T_span → T_generalInterval |
|  | *B_intersection:* | T_generalInterval → T_generalInterval |
|  | *B_expand:* | T_span → T_generalInterval |
|  | *B_shrink:* | T_span → T_generalInterval |
|  | *B_meets:* | T_generalInterval → T_boolean |
|  | *B_lowerBound:* | T_instant |
|  | *B_upperBound:* | T_instant |
|  | *B_lowerBoundOpen:* | T_boolean |
|  | *B_upperBoundOpen:* | T_boolean |

Table 4: Behaviors defined on general time intervals.

`T_generalInterval` defines behaviors *B_lowerBoundOpen* and *B_upperBoundOpen* to model different kinds of open, closed, half open and half closed intervals. Behavior *B_lowerBoundOpen* (*B_upperBoundOpen*) returns a boolean value which shows whether the lower bound (upper bound) of a time interval is open or closed. As an illustration of why different kinds of intervals are needed, consider the example of a reporter (employed by a news agency covering a murder trial) who enters an object prosecutionSpeech$_1$ in the objectbase at time $18:00\ Jan\ 25$ and corrects the text of the speech to prosecutionSpeech$_2$ at time $15:00\ Jan\ 27$. The history of these transactions can be represented as $\{\langle[18:00\ Jan\ 25,\ 15:00\ Jan\ 27), \text{prosecutionSpeech}_1\rangle, \langle[15:00\ Jan\ 27,\ now], \text{prosecutionSpeech}_2\rangle\}$. The time *now* represents that the object prosecutionSpeech$_2$ is currently valid. The upper bound of the interval $[18:00\ Jan\ 25,\ 15:00\ Jan\ 27)$ should be open, otherwise the objectbase will be inconsistent since at time $15:00\ Jan\ 27$ both prosecutionSpeech$_1$ and prosecutionSpeech$_2$ will be present in the objectbase. The lower bound of the interval $[15:00\ Jan\ 27,\ now]$ should be closed, otherwise there will be no information at time $15:00\ Jan\ 27$. Its upper bound should also be closed or there will be no information available currently, and the objectbase will not be up to date. This example illustrates the need for the [] and [) time intervals. The remaining two kinds of intervals (i.e., () and (]) come as a result of the combination of the *B_lbOpen* and *B_ubOpen* behaviors and have been added for completeness.

`T_generalInterval` defines additional unary behaviors *B_lowerBound* and *B_upperBound* which return the lower bound, upper bound and duration of a time interval, respectively.

`T_generalInterval` inherits all the interval comparison behaviors from `T_timeSet` and defines an additional comparison behavior, *B_meets*. The behavior *B_meets* models the *meets* operation between intervals as shown in Figure 2.

The set-theoretic behaviors are inherited from `T_timeSet`. *B_intersection* is refined to return a general interval as the intersection between two general intervals. The union of two disjoint intervals causes *B_union* to return an object of type `T_null`[5], that is, an error occurs. For the difference operation, if the second interval is contained in the first, an error occurs – that is, *B_difference* returns an object of type `T_null`. However, if the first interval is contained in the second, *B_difference* returns the null interval. In the intersection operation, if the two intervals are disjoint, *B_intersection* returns the null interval. The null interval is the only instance of the `T_specialInterval` type. `T_specialInterval` is a subtype of all interval types as discussed later on. This effectively means that the `T_generalInterval` type is closed under the set-theoretic operations (*union*, *difference*, and *intersection*) since every instance of `T_null` or `T_specialInterval` is also an instance of `T_generalInterval`. This follows from the fact that `T_null` and `T_specialInterval` are subtypes of `T_generalInterval` as shown in Figure 8.

Behaviors *B_add* and *B_subtract* are refined in `T_generalInterval` to return an object of type `T_generalInterval` as a result. The *B_add* behavior in `T_generalInterval` is a sort of displacement behavior which displaces the time interval by a duration of time. For example, $[5\ January\ 1987, 10\ January\ 1987)$·*B_add*($5\ days$) returns the time interval $[10\ January\ 1987, 15\ January\ 1987)$ and $[5\ January\ 1987 \sim 10\ January\ 1987)$·*B_add*($5\ days$) returns the time interval $[10\ January\ 1987 \sim 15\ January\ 1987)$. The set-theoratic intersection behavior is also refined to return the intersection between two general intervals. `T_generalInterval` also defines new behaviors. Behavior *B_expand* expands a time interval by a certain time duration. For example, $[5\ January\ 1987, 10\ January\ 1987)$·*B_expand*($5\ days$) returns the time interval $[5\ January\ 1987, 15\ January\ 1987)$ and $[5\ January\ 1987 \sim 10\ January\ 1987)$·*B_expand*(5) returns the time interval $[5\ January\ 1987 \sim 15\ January\ 1987)$. Behavior *B_shrink* shrinks a time interval by a certain time duration. If the time duration by which an interval is to be shrunk is greater than the length of the interval, then an error occurs, i.e., an object of type `T_null` is returned. For example, $[5\ January\ 1987, 10\ January\ 1987)$·*B_shrink*($7\ days$) returns an object of type `T_null` (an error).

We define `T_determinateInterval` and `T_indeterminateInterval` as subtypes of `T_generalInterval`. These two types correspond to the notions of continuous determinate and indeterminate intervals, respectively. Likewise, we introduce `T_discreteDeterminateInterval` and `T_discreteIndeterminateInterval` as subtypes of `T_generalInterval` to model two different kinds of discrete intervals present in our model. Behaviors on these types are inherited from `T_generalInterval` and refined accordingly as shown in Table 5. There is also one additional

[5]The `T_null` type is a subtype of all other types in the TIGUKAT type lattice as shown in Figure 1.

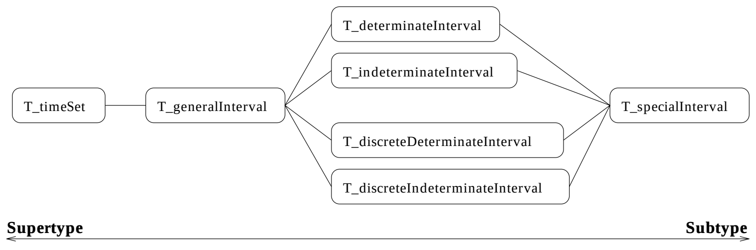

Figure 5: Interval types.

interval type `T_specialInterval` defined as a subtype of the `T_determinateInterval`, `T_indeterminateInterval`, `T_discreteDeterminateInterval` and `T_discreteIndeterminateInterval` types. The `T_specialInterval` type has only one instance which represents the *null* interval. The *B_lowerBound* and *B_upperBound* behaviors on the null interval return an object of `T_null`, i.e., an error. The partial interval time type hierarchy is shown in Figure 5.

Since different intervals have many common properties (operations, behaviors) we define a set of *abstract types* to model these common features. These types include `T_generalDeterminateInterval`, `T_generalIndeterminateInterval`, `T_interval`, and `T_discreteInterval`[6]. The latter two abstract types define behaviors common to continuous and discrete intervals, respectively. These types and their corresponding behaviors are shown in Table 6. To keep the type hierarchy simple for illustration purposes, the set of abstract types (except `T_generalInterval`) are not shown in Figure 5. Instead, they are shown in Figure 8 which gives the complete time type hierarchy.

## 4.4 Time Instants

In our framework, every instant can be treated as an interval with identical lower and upper bounds. As discussed in Section 3.2.2, discrete time instants essentially behave like continuous time intervals. More specifically, a discrete time instant $t_{discr}$ with a granularity $G_t$ is a continuous time interval of the form $[t_{cont}, t_{cont} + G_t)$ ($[t_{cont} \sim t_{cont} + G_t)$ if $t_{discr}$ is indeterminate). Additionally, a discrete time instant is also a special kind of discrete time interval whose lower and upper bounds are discrete and identical. Therefore we define `T_discreteDeterminateInstant` as a subtype of `T_discreteDeterminateInterval` and `T_determinateInterval`. Similarly, `T_discreteIndeterminateInstant` is defined as a subtype of `T_discreteIndeterminateInterval` and `T_indeterminateInterval`. The `T_instant` models continuous instants and is a subtype of

[6]Note that the set of abstract types also includes `T_generalInterval` that defines behaviors common for all intervals as discussed earlier. Behaviors on `T_generalInterval` are given in Table 4.

| | | |
|---|---|---|
| T_determinateInterval | *B_add:* | T_span → T_determinateInterval |
| | *B_subtract:* | T_span → T_determinateInterval |
| | *B_intersection:* | T_generalDeterminateInterval → T_determinateInterval |
| | *B_expand:* | T_span → T_determinateInterval |
| | *B_shrink:* | T_span → T_determinateInterval |
| T_indeterminateInterval | *B_add:* | T_span → T_indeterminateInterval |
| | *B_subtract:* | T_span → T_indeterminateInterval |
| | *B_intersection:* | T_generalInterval → T_indeterminateInterval |
| | *B_difference:* | T_generalIndeterminateInterval → T_indeterminateInterval |
| | *B_expand:* | T_span → T_indeterminateInterval |
| | *B_shrink:* | T_span → T_indeterminateInterval |
| T_discreteDeterminateInterval | *B_add:* | T_discreteSpan → T_discreteDeterminateInterval |
| | *B_subtract:* | T_discreteSpan → T_discreteDeterminateInterval |
| | *B_intersection:* | T_interval → T_interval |
| | | T_discreteDeterminateInterval → T_discreteDeterminateInterval |
| | *B_expand:* | T_discreteSpan → T_discreteDeterminateInterval |
| | | T_discreteIndeterminateSpan → T_discreteTimeSet |
| | *B_shrink:* | T_discreteSpan → T_discreteDeterminateInterval |
| | | T_discreteIndeterminateSpan → T_discreteTimeSet |
| | *B_length:* | T_discreteSpan |
| | *B_lowerBound:* | T_discreteDeterminateInstant |
| | *B_upperBound:* | T_discreteDeterminateInstant |
| T_discreteIndeterminateInterval | *B_add:* | T_discreteSpan → T_discreteIndeterminateInterval |
| | *B_subtract:* | T_discreteSpan → T_discreteIndeterminateInterval |
| | *B_intersection:* | T_discreteInterval → T_discreteIndeterminateInterval |
| | *B_difference:* | T_generalIndeterminateInterval → T_discreteIndeterminateInterval |
| | *B_expand:* | T_discreteSpan → T_discreteIndeterminateInterval |
| | *B_shrink:* | T_discreteSpan → T_discreteIndeterminateInterval |
| | *B_discreteLowerBound:* | T_discreteIndeterminateInstant |
| | *B_discreteUpperBound:* | T_discreteIndeterminateInstant |

Table 5: Behaviors defined on concrete time intervals.

both `T_determinateInterval` and `T_indeterminateInterval`. Note that just one type is needed for the continuous time instants because the period of indeterminacy for a continuous time instant is zero. That is, continuous indeterminate time instants do not exist. This is also the reason why discrete instant types are not subtypes of `T_instant` but rather those of `T_determinateInterval` and `T_indeterminateInterval`, respectively. As a consequence, `T_discreteInterval` is not a subtype of `T_interval` as the former has discrete instants as bounds while the latter has continuous instants as shown in Figure 8. There is also one additional instant type `T_specialInstant` defined as a subtype of all instant types. The only instances of this type are $+\infty$ and $-\infty$. These instances belong simultaneously to discrete time domains of all granularities as well as to the continuous time domain. The partial time instant type hierarchy is shown in Figure 6.

To encompass the behaviors common to different kinds of instants we introduce a set of abstract

| | | |
|---|---|---|
| T_interval | *B_add:* | T_span → T_interval |
| | *B_subtract:* | T_span → T_interval |
| | *B_intersection:* | T_interval → T_interval |
| | *B_expand:* | T_span → T_interval |
| | *B_shrink:* | T_span → T_interval |
| T_discreteInterval | *B_add:* | T_discreteSpan → T_discreteInterval |
| | *B_subtract:* | T_discreteSpan → T_discreteInterval |
| | *B_intersection:* | T_discreteInterval → T_discreteInterval |
| | *B_expand:* | T_discreteSpan → T_discreteInterval |
| | *B_shrink:* | T_discreteSpan → T_discreteInterval |
| | *B_discreteLowerBound:* | T_discreteInstant |
| | *B_discreteUpperBound:* | T_discreteInstant |
| T_generalDeterminateInterval | *B_add:* | T_span → T_generalDeterminateInterval |
| | *B_subtract:* | T_span → T_generalDeterminateInterval |
| | *B_intersection:* | T_generalDeterminateInterval → T_generalDeterminateInterval |
| | *B_expand:* | T_indeterminateSpan → T_timeSet<br>T_span → T_generalDeterminateInterval |
| | *B_shrink:* | T_indeterminateSpan → T_timeSet<br>T_span → T_generalDeterminateInterval |
| | *B_length:* | T_span |
| T_generalIndeterminateInterval | *B_add:* | T_span → T_generalIndeterminateInterval |
| | *B_subtract:* | T_span → T_generalIndeterminateInterval |
| | *B_intersection:* | T_generalInterval → T_generalIndeterminateInterval<br>T_interval → T_indeterminateInterval |
| | *B_difference:* | T_generalIndeterminateInterval → T_generalIndeterminateInterval |
| | *B_expand:* | T_span → T_generalIndeterminateInterval |
| | *B_shrink:* | T_span → T_generalIndeterminateInterval |

Table 6: Behaviors defined on abstract time intervals.

types which includes `T_generalInstant`, `T_generalDeterminateInstant`, `T_generalIndeterminateInstant`, and `T_discreteInstant`. These types are analogous to the appropriate abstract interval types described earlier. The types and behaviors defined on time instants are shown in Table 7. To keep the type hierarchy simple for illustration purposes, the set of instant abstract types are not shown in Figure 6. Instead, they are shown in Figure 8 which gives the complete time type hierarchy. The *B_succ* and *B_pred* behaviors in `T_discreteInstant` return the next and previous time instant of a discrete time instant, respectively. *B_granularity* returns the granularity of a time instant. This granularity is the logical granularity of a time instant and not its physical one. For example, 5 *January* 1987·*B_granularity* could return $G_{hours}$ even though the time instant 5 *January* 1987 appears to have a granularity of $G_{days}$. Native behaviors defined in `T_instant` are the comparison behaviors *B_lessthaneqto* and *B_greaterthaneqto* (these are essentially the $\leq$ and $\geq$ operators, respectively), and the *B_elapsed* behavior which returns the elapsed time (duration) between two

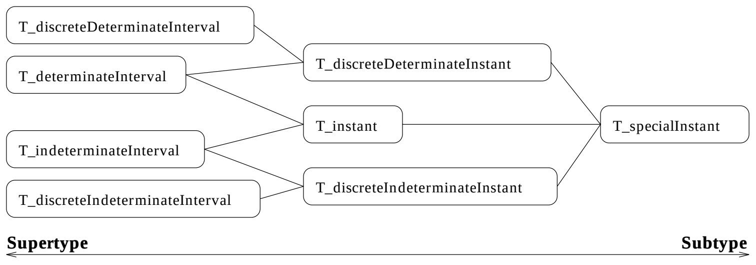
T_discreteDeterminateInterval
T_determinateInterval
T_indeterminateInterval
T_discreteIndeterminateInterval
T_discreteDeterminateInstant
T_instant
T_discreteIndeterminateInstant
T_specialInstant
Supertype
Subtype

Figure 6: Instant types.

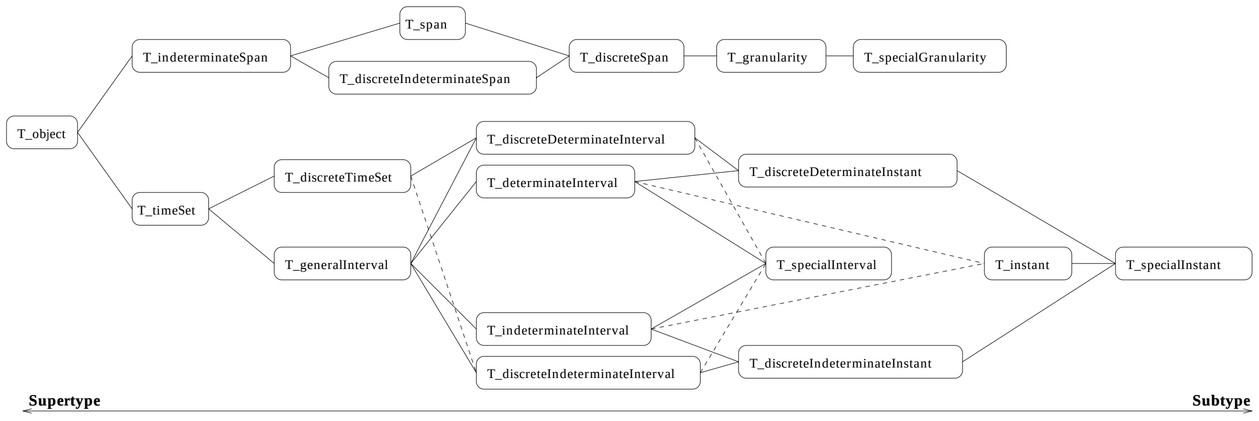

Figure 7: The time type hierarchy.

time instants.

Figure 7 ties together the partial time type hierarchies given in Figures 3-6, and gives a more complete picture of the hierarchy of the time types introduced so far.

## 4.5 Other Relevant Types

So far we have described only temporal types. However, there are types that do not belong to the type system of the temporal model but are related to it in one way or another. These types include `T_object`, `T_discrete`, and `T_atomic`. The type `T_object` is the single root of the TIGUKAT type lattice and is therefore a supertype of every type in the system including the types defined in the temporal model. It defines the behaviors common to all objects (such as *B_self*). `T_discrete` defines the "successor" and "predecessor" behaviors common to all discrete domains. Therefore it is a supertype of both `T_discreteSpan` and `T_discreteInstant`. The type `T_atomic` is an abstract supertype of all types of atomic objects. Since time instants and determinate time spans are considered atomic in our model, `T_span` and `T_generalInstant` are defined as subtypes of `T_atomic`. The complete time time hierarchy with all abstract and concrete types is given in

| | | |
|---|---|---|
| T_generalInstant | *B_add:* | T_span → T_generalInstant |
| | *B_subtract:* | T_span → T_generalInstant |
| | *B_intersection:* | T_generalInstant → T_generalInstant |
| T_generalDeterminateInstant | *B_intersection:* | T_generalDeterminateInstant → T_generalDeterminateInstant |
| T_generalIndeterminateInstant | *B_intersection:* | T_generalInstant → T_generalIndeterminateInstant |
| | *B_difference:* | T_generalIndeterminateInterval → T_generalIndeterminateInstant |
| T_discreteInstant | *B_intersection:* | T_discreteInstant → T_discreteInstant |
| | *B_succ:* | T_discreteInstant |
| | *B_pred:* | T_discreteInstant |
| | *B_granularity:* | T_granularity |
| T_discreteDeterminateInstant | *B_intersection:* | T_discreteDeterminateInstant → T_discreteDeterminateInstant |
| | *B_succ:* | T_discreteDeterminateInstant |
| | *B_pred:* | T_discreteDeterminateInstant |
| T_discreteIndeterminateInstant | *B_intersection:* | T_discreteIndeterminateInstant → T_discreteIndeterminateInstant |
| | *B_difference:* | T_discreteIndeterminateInterval → T_discreteIndeterminateInstant |
| | *B_succ:* | T_discreteIndeterminateInstant |
| | *B_pred:* | T_discreteIndeterminateInstant |
| T_instant | *B_add:* | T_span → T_instant |
| | *B_subtract:* | T_span → T_instant |
| | *B_intersection:* | T_timeSet → T_instant |
| | *B_difference:* | T_timeSet → T_instant |
| | *B_leq:* | T_instant → T_boolean |
| | *B_geq:* | T_instant → T_boolean |
| | *B_elapsed:* | T_instant → T_span |

Table 7: Behaviors defined on time instants.

Figure 8.

## 5 Related Work

Most of the research on temporal databases has concentrated on extending the relational model to handle time in an appropriate manner. These extensions can be divided into two main categories. The first approach uses normalized (1NF) relations in which special time attributes are added (called *tuple time-stamping*) and the history of an object (attribute) is modeled by several 1NF tuples [LJ88, Sno87]. The second approach uses non-normalized (N1NF) relations and attaches time to attribute values (called *attribute time-stamping*) in which the history of an object is modeled by a single N1NF tuple [Gad88, Tan86]. 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 objectbase management systems. 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.

Although there have been a substantial number of proposals on temporal object models (e.g.,

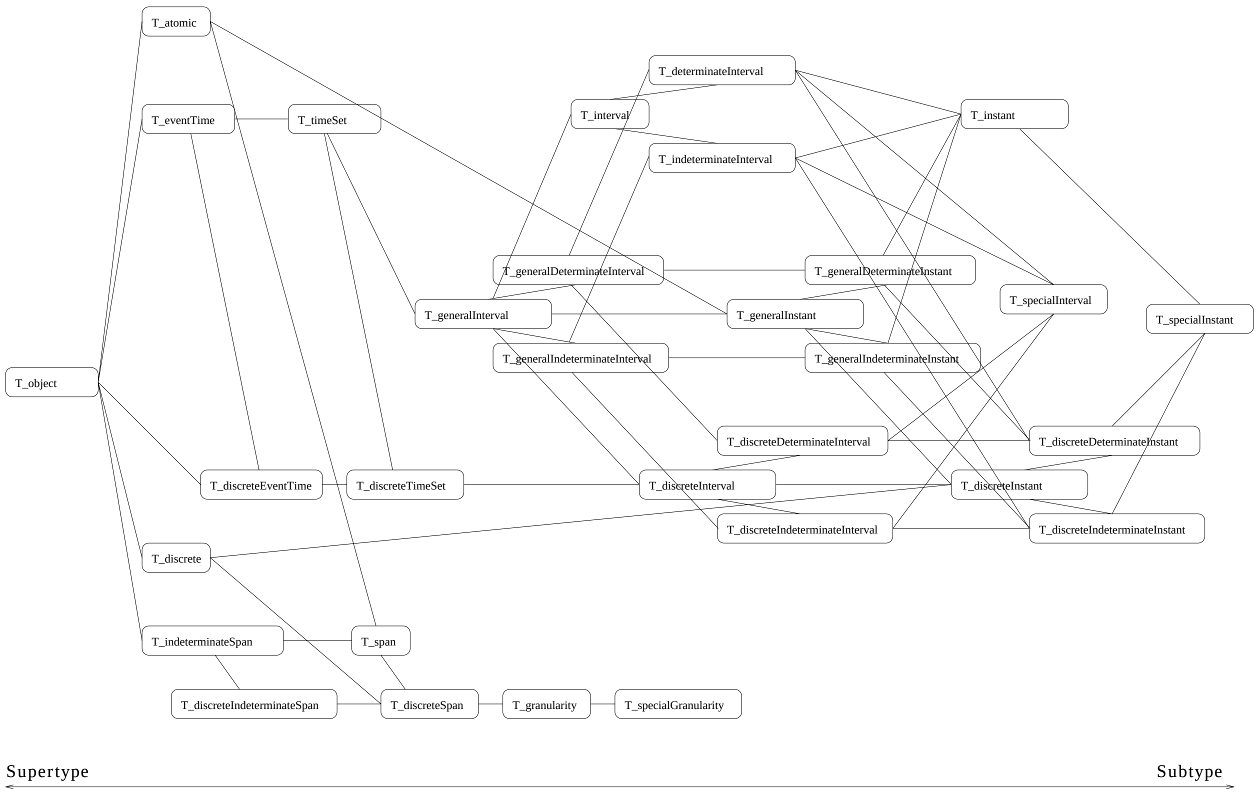

Figure 8: The complete time type hierarchy.

[KC86, RS91, RS93, KS92, WD92, DW92, SC93, CG93]), none of them provides support for handling both the continuous and discrete time domains, multiple granularities and indeterminate temporal information.

In the context of OBMSs, [KC86] describes a model to handle complex objects and discusses the representation and temporal dimensions to support object identity. However, most of the emphasis is on the representation dimension and it is not clear how temporal primitives and different domains of time are supported.

An extension to an object-based entity-relationship model to incorporate temporal structures and constraints in the data model is given in [RS91]. A corresponding temporal object-oriented algebra is given in [RS93]. The temporal model is quite structural and closely tied to the relational model. Furthermore, a single domain of time is supported.

In [KS92], a state of a complex object is represented by the notion of a *time slice* which basically consists of a time interval and the object which was valid during the interval. It is not clear however, how other timestamps and domains of time are supported for different applications.

In [WD92, DW92], variables and quantifiers are used to range over time. This work is based on abstract notions of time and abstract time types are used (similar to ours) to facilitate the modeling of various notions of time. However, the placement of these abstract types in the type lattice is not precisely specified. The behaviors on these abstract types are not defined either. Essentially, the user (or database designer) is given the responsibility of defining most of the temporal model, which includes specifying the temporal schema and specifying the queries. We contend that specifying a schema for modeling time for an application such as *HotNews*, which has diverse requirements, is not trivial and 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 requirements of *HotNews* can uniformly and consistently be supported in our model.

In OSAM*/T [SC93], object histories are linear and are recorded discretely by time interval stamps. There is no support for the continuous time domain, time instants, or time spans.

Cheng and Gadia [CG93] have proposed an object-oriented model which captures the semantics of temporal objects through type inheritance. However, no clear semantics has been given for different timestamps and time domains.

Most of the research on temporal relational models has concentrated on modeling determinate temporal information with a single underlying granularity. There have been some recent proposals however, that handle multiple granularities.

Clifford and Rao [CR87] introduce a general structure for time domains called a *temporal universe*. A temporal universe consists of a totally ordered set of granularities. Operations are defined on a temporal universe, which basically convert different times to a (common) finer granularity before carrying out the operation. Our work is more general in that we allow conversion from finer to coarser granularities and vice-versa as per user needs. We also do not require that operands be converted to a common granularity before an operation. More specifically, we allow operations between mixed granularities.

Wiederhold et al., [WJL91] also examine the issue of multiple granularities. An algebra is described that allows the conversion of event times to an interval representation. This usually involves converting the coarser granularity to the finer granularity in light of the semantics of the time varying domains. Our work encompasses this approach in that we are able to convert coarser granularities to finer ones and vice-versa using temporal indeterminacy. [WJS93] introduces *temporal modules* and *extended temporal modules*. These modules use windowing functions which are of different time units to provide access to temporal relations. Additionally, they provide semantics for moving up and down a granularity lattice. Once again, our work encompasses their approach in that we show how we can convert one granularity to another, and how this leads to temporal indeterminacy.

In [BP85] the issues of absolute, relative, imprecise and periodic times are discussed. Multiple granularities are supported for each time. Operands in operations involving mixed granularities are converted to the coarser granularity to avoid indeterminacy. In a more recent work [MPB92], the existence of a minimum underlying granularity (quantum of time) to which time is mapped, is assumed. Furthermore, as in [CR87], time spans are converted to the same time metric and operations are performed at the common lowest level of the operands. In our model, we do not need to assume the existence of a minimum underlying granularity since we have the notion of continuous time. Moreover, we allow operations between mixed granularities and hence do not need to convert time spans to a common lowest granularity before performing operations.

Gadia et al., [GNP92] combine value[7] and temporal indeterminacy. In this paper, we deal only with temporal indeterminacy. We contend that value and temporal indeterminacy are orthogonal and the former can easily be incorporated within the history representation of our temporal object model. The model presented in [GNP92] is discrete with a single underlying granularity. In our approach, we allow for both discrete and continuous time domains with multiple granularities. We also show how temporal indeterminacy results from granularities.

In [Kou93] temporal tables are used as syntactic devices for the representation of possibly imprecise temporal knowledge. Temporal tables can have variables as attribute values which represent values that are not completely known, but for which a global temporal constraint holds. In this work we do not consider temporal constraints. Rather, we focus on modeling temporal indeterminacy at the time primitives level namely, spans, instants, and intervals. Furthermore, we uniformly accommodate temporal indeterminacy arising from the representation of multiple granularities.

Our work is closest to that of Dyreson & Snodgrass [DS93] in that they support multiple granularities and deal with temporal indeterminacy. However, there are some differences between our approach and theirs:

- We consistently represent discrete time information with different granularities and continuous time information in the same framework and show how one can be converted to the other consistently. They deal with only discrete temporal indeterminate information.

- We contend that there should be two interpretations for an event that takes place at a certain

[7]Value indeterminacy is where the value of an attribute is not fully known.

time instant like $8pm\ Jan$ 12. The event could be interpreted as having occurred sometime during $8pm$ to $9pm$, or, as having lasted the entire one hour duration. In our framework, both interpretations are modeled, whereas in theirs only the first is represented.

- They have a single underlying granularity which is system defined (chronon). The correctness of their theory depends on the existence of such a granularity. We do not make such an assumption; this allows us to seamlessly incorporate continuous time in our model.

- Before every binary operation they convert both operands to a common granularity. Our approach allows us to avoid these conversions, thereby preventing information loss. For example, they cannot represent time spans like 1 *month and* 1 *day*. This would have to be represented as days (assuming the underlying base granularity is a day), necessarily leading to an indeterminate span and thereby losing information. Additionally, consider adding a time span of 1 *month* to the time instant 1 *Februray* 1995. In our framework, this will result in the time instant 1 *March* 1995. However, in their approach a conversion has to take place leading to information loss and an indeterminacy of 3 days. The margin of indeterminacy increases if a longer time span is added. For example, adding a time span of 10 *months* will result in an indeterminacy of 1 month.

- While adding indeterminate time spans to determinate time instants, they lose the duration information. We prevent this by using event times.

- We provide a clear set-theoretic algebra for interpreting our anchored time (both determinate and indeterminate) specifications. In their work, an algebra is not clearly defined.

## 6 Conclusions and Future Work

In this work we have defined a temporal object model by providing an extensible set of primitive time types with a rich set of behaviors to model various notions of time elegantly. Our model supports time instants, time intervals and time spans. We showed how both the discrete and continuous domains of time are modeled.

We have also provided a framework which uniformly handles different granularities of time and indeterminate temporal information within a behavioral temporal object model. We have shown how we can consistently represent discrete temporal information with different granularities and continuous time within the same framework. We also described how different granularities can be converted to each other. Additionally, we showed how we can represent and operate on indeterminate time spans, intervals and instants. Furthermore, we showed how determinate and indeterminate instants and intervals are mapped to a set-theoretic framework.

Our work allows new user-defined granularities within the set of existing granularities. For example, the user may want to add a new granularity of 10 days. The system should be able to uniformly and consistently incorporate this granularity within the existing framework. We are also looking at dealing with periodic temporal information, e.g., "a temporal workshop occurs every year," and incorporating different calendric systems in the temporal model. Furthermore, we are looking at the concept of *event times*. Event times allow us to maintain the lengths of the events in addition to their start and end times. This allows us to represent events like "the murder took place on $Jan$ 12 and lasted for 4 minutes."

Currently, we are working on incorporating different time orders (linear and branching) into our model. We are also extending the model to incorporate both valid and transaction time histories [GLÖS95]. Additionally, we are looking at storing and operating on times that have not been bound to any concrete value, for example the time *now*.

We are also working on temporal extensions to the TIGUKAT algebra [Pet94] which will form the basis of a temporal query language as an extension of the TIGUKAT query language TQL [PLÖS93]. This language will enable us to access and manipulate temporal objects, answer historical queries, prepare summaries of past performance, or make projections into the future.

## References

[All84] J. F. Allen. Towards a General Theory of Action and Time. *Artificial Intelligence*, 23(123), 1984.

[BP85] F. Barbic and B. Pernici. Time Modeling in Office Information Systems. In *Proc. ACM SIGMOD Int'l. Conf. on Management of Data*, pages 51–62, Austin, Texas, May 1985.

[CG93] 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.

[Cho94] J. Chomicki. Temporal Query Languages: a Survey. In *Proceedings of the International Conference on Temporal Logic*, Bonn, Germany, July 1994.

[CR87] J. Clifford and A. Rao. A Simple, General Structure for Temporal Domains. In C. Rolland, F. Bodart, and M. Leonard, editors, *Temporal Aspects in Information Systems*, pages 17–30. North-Holland, 1987.

[DS93] C.E. Dyreson and R.T. Snodgrass. Valid-time Indeterminacy. In *Proc. 9th Int'l. Conf. on Data Engineering*, pages 335–343, April 1993.

[DSS94] C.E. Dyreson, M.D. Soo, and R.T. Snodgrass. The TSQL2 Data Model for Time. A TSQL Commentary. September 1994.

[DW92] 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.

[Gad88] S. Gadia. A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on Database Systems*, 13(4), 1988.

[GLÖS95] I.A. Goralwalla, Y. Leontiev, M.T. Özsu, and D. Szafron. Timelines, Histories, Event Times and Calenders in a Uniform Behavioral Temporal Object Model. Technical report, University of Alberta, 1995. In press.

[GNP92] S.K. Gadia, S. Nair, and Y-C. Poon. Incomplete Information in Relational Temporal Databases. In *Proc. 18th Int'l Conf. on Very Large Data Bases*, pages 395–406, August 1992.

[KC86] 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.

[Kou93] M. Koubarakis. Representation and Querying in Temporal Databases: the Power of Temporal Constraints. In *Proc. 9th Int'l. Conf. on Data Engineering*, pages 327–334, April 1993.

[KS92] 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.

[LJ88] N. Lorentzos and R. Johnson. Extending Relational Algebra to Manipulate Temporal Data. *Information Systems*, 15(3), 1988.

[MPB92] R. Maiocchi, B. Pernici, and F. Barbic. Automatic Deduction of Temporal Information. *ACM Transactions on Database Systems*, 17(4):647–688, 1992.

[ÖPS+95] 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*, 1995. In press.

[ÖSEV95] M.T. Özsu, D. Szafron, G. ElMedani, and C. Vittal. An Object-Oriented Multimedia Database for News-on-Demand Application. *Multimedia Systems*, 1995. In press.

[Pet94] R.J. Peters. *TIGUKAT: A Uniform Behavioral Objectbase Management System.* PhD thesis, University of Alberta, 1994.

[PLÖS93] 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.

[PÖ93] R.J. Peters and M.T. Özsu. Reflection in a Uniform Behavioral Object Model. In *Proc. 12th Int'l Conf. on the Entity Relationship Approach*, pages 37–49, December 1993.

[RS91] 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.

[RS93] E. Rose and A. Segev. TOOA - A Temporal Object-Oriented Algebra. In *Proc. European Conference on Object-Oriented Programming*, 1993.

[SC93] 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.

[Sno87] R. Snodgrass. The Temporal Query Language, TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[Sno92] R.T. Snodgrass. Temporal Databases. In *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, pages 22–64. Springer-Verlag, LNCS 639, 1992.

[Tan86] A. Tansel. Adding Time Dimension to Relational Model and Extending Relational Algebra. *Information Systems*, 13(4):343–355, 1986.

[WD92] 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.

[WJL91] G. Wiederhold, S. Jajodia, and W. Litwin. Dealing with Granularity of Time in Temporal Databases. In R. Andersen, J.A. Bubenko Jr., and A. Solvberg, editors, *Advanced Information Systems Engineering, 3rd Int'l Conference CAiSE '91*, pages 124–140. Springer-Verlag, 1991.

[WJS93] X. Wang, S. Jajodia, and V. Subrahmanian. Temporal Modules: An Approach Toward Temporal Databases. In *Proc. ACM SIGMOD Int'l. Conf. on Management of Data*, pages 227–236, 1993.