A Framework for Temporal Data Models: Exploiting Object-Oriented Technology*

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

Most of the database research on modeling time has concentrated on the definition of a particular temporal model and its incorporation into a (relational or object) database management system. This has resulted in quite a large number of different temporal models, each providing a specific set of temporal features. This paper presents an object-oriented framework for temporal models which supports multiple notions of time. The framework can be used to accommodate the temporal needs of different applications, and derive existing temporal models by making a series of design decisions through subclass specialization. It can also be used to derive a series of new more general temporal models that meet the needs of a growing number of emerging applications.

1 Introduction

Time is an attribute of all real-world phenomena. Events occur at specific points in time; objects and the relationships among objects exist over time. The ability to model the temporal dimension of the real world is essential for many applications such as econometrics, banking, inventory control, medical records, real-time systems, multimedia, airline reservations, versions in CAD/CAM applications, statistical and scientific applications, etc. Database management systems (DBMSs) that support these applications have to be able to satisfy temporal requirements.

In order to accommodate the temporal needs of different applications, there has been extensive research activity on temporal data models in the last decade [18, 23, 22, 12, 25]. The early research concentrated on extending the relational data model to handle time in an appropriate manner. The notion of time, with its multiple facets, is difficult (if not impossible) to represent in one single relational model since it does not adequately capture data or application semantics. This is substantiated by most of the relational temporal models only supporting a model of time that is discrete and linear.

The general limitation of the relational model in supporting complex applications has led to research into next-generation data models, specifically object data models. The research on temporal models has generally followed this trend. Temporal object models can more accurately capture the semantics of complex objects and treat time as a basic component. There have been many temporal object model proposals (for example, [16, 24, 26, 11, 5, 3]). None of these models, with the exception of [26], reflects on the capability of an object model in defining the diversity of time. Wuu & Dayal [26] provide an abstract *time* type to model the most general semantics of time which can then be subtyped (by the user or database designer) to model the various notions of time required by specific applications. However, this requires significant support from the user, including specification of the temporal schema.

Both (relational and object-oriented) approaches have led to the definition and design of a multitude of temporal models. Many of these assume a set of fixed notions about time, and therefore do not incorporate sufficient functionality or extensibility to meet the varying temporal requirements of today's applications. Instead, similar functionality is re-engineered every time a temporal model is created for a new application.

Although most temporal models were designed to support the temporal needs of a particular application, or group of similar applications, if we look at the functionality offered by the temporal models at an abstract level, there are notable similarities in their temporal features:

- Each temporal model has one or more temporal primitives, namely, *time instant*, *time interval*, *time span*, etc. The *discrete* or the *continuous* domain is used by each temporal model as a temporal domain over the primitives.

- Some temporal models require their temporal primitives to have the same underlying *granularity*, while others support multiple granularities and allow temporal primitives to be specified in different granularities.

- Most temporal models support a *linear* model of time, while some support a *branching* model of time. In the former, temporal primitives are totally ordered, while in the latter they have a partial order defined on them.

- All temporal models provide some means of modeling historical information about real-world entities and/or histories of entities in the database. Two of the most popular types of histories that have been employed are *valid* and *transaction* time histories [21], respectively.

These commonalities suggest a need for combining the diverse features of the temporal domain under a single infrastructure that allows design reuse. In this paper, we present an object-oriented framework [10] that provides such a unified infrastructure. An object-oriented approach allows us to capture the complex semantics of time by representing it as a basic entity. Furthermore, the typing and inheritance mechanisms of object-oriented systems directly enable the various notions of time to be reflected in a single framework.

We can draw a parallel between our work and similar (albeit on a much larger scale) approaches used in *Choices* [4] and *cmcc* [1]. *Choices* is a framework for operating system construction which was designed to provide a family of operating systems that could be reconfigured to meet diverse user/application requirements. *cmcc* is an optimizing compiler that makes use of frameworks to facilitate code reuse for different modules of a compiler. Similar to *Choices* and *cmcc*, the temporal framework can be regarded as an attempt to construct a family of temporal models. The framework can then be tailored to reflect a particular temporal model which best suits the needs of an application. A particular temporal model would be one of the many "instances" of the framework.

The presentation of this paper is divided into four sections. Section 2 presents the temporal framework by identifying the design dimensions (key abstractions) for temporal models and the interactions between them. Section 3 illustrates how the temporal framework can be tailored to accommodate the temporal needs of different applications, and the temporal features of temporal object models. Section 4 summarizes the work presented in this paper and outlines avenues for future research.

2 The Architecture of the Temporal Framework

In order to accommodate the varying requirements that many applications have for temporal support, we first identify the design dimensions that span the design space for temporal models. Next, we identify the components or features of each design dimension. Finally, we explore the interactions between the design dimensions in order to structure the design space. These steps produce a framework which consists of abstract and concrete object types, and properties. The types are used to model the different design dimensions and their corresponding components. The properties are used to model the different operations on each component, and to represent the relationships (constraints) between the design dimensions. The framework classifies design alternatives for temporal models by providing types and properties that can be used to define the semantics of many different specific notions of time.

2.1 Design Dimensions

The design alternatives for temporal models can be classified along four design dimensions:

1. Temporal Structure
2. Temporal Representation
3. Temporal Order
4. Temporal History

These design dimensions span the design space of temporal models. Temporal structures provide the underlying ontology and domains for time. A temporal representation provides a means to represent time so that it is human readable. Temporal orders give an ordering to time. Temporal histories allow events and activities to be associated with time. In the rest of this section, we describe each design dimension in detail.

We assume the availability of commonly used object-oriented features – *atomic entities* (reals, integers, strings, etc.); *types* for defining common features of objects; *properties* (which represent *methods* and *instance variables*) for specifying the semantics of operations that may be performed on objects; *classes* which represent the extents of types; and *collections* for supporting general heterogeneous groupings of objects. In this paper, a reference prefixed by "T_" refers to a type, and "*P_*" to a property. A type is represented by a rounded box. An abstract type is shaded with a black triangle in its upper left corner, while a concrete type is unshaded. In Figures 5 and 12 the rectangular boxes are objects. Objects have an outgoing edge

for each property applicable to the object which is labeled with the name of the property and which leads to an object resulting from the application of the property to the given object. A circle labeled with the symbols $\{\ \}$ represents a container object and has outgoing edges labeled with "$\in$" to each member object.

2.1.1 Temporal Structure

The first question about a temporal model is "what is its underlying temporal structure?" More specifically, what are the temporal primitives supported in the model, what temporal domains are available over these primitives, and what is the temporal determinacy of the primitives? Indeed, the temporal structure dimension with its various constituents forms the basic building block of the design space of any temporal model since it is comprised of the basic temporal features that underlie the model.

Figure 1 shows the building block hierarchy of a temporal structure. The basic building block consists of anchored and unanchored temporal primitives. The next building block provides a domain for the primitives that consists of discrete or continuous temporal primitives. Finally, the last building block of Figure 1 adds determinacy. Thus, a temporal structure can be defined by a series of progressively enhanced temporal primitives.

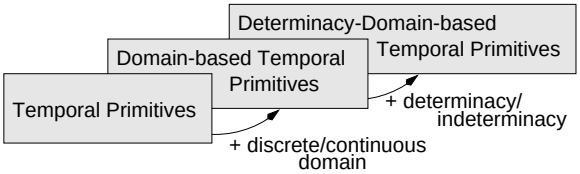

Figure 1: Building a Temporal Structure

Figure 2 gives a detailed hierarchy of the different types of temporal primitives that exist in each of the building blocks of Figure 1. Based on the features of a temporal structure, its design space consists of 11 different kinds of temporal primitives. These are the determinacy-domain-based temporal primitives shown in Figure 2 and described below.

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." Since in theory, continuous instants have infinite precision, they cannot have a period of indeterminacy. Therefore, continuous indeterminate time instants do not exist in Figure 2.

However, continuous intervals can be determinate or indeterminate. The difference between them is that a continuous determinate interval denotes an event that occurred during each instant of the interval whereas a continuous indeterminate interval denotes an event that occurs at one or more instants of the interval. Every interval has lower and upper bounds which are continuous instants.

Discrete time instants and intervals Assume that somebody has been on a train the whole day of January 5th, 1987. This fact can be expressed using a determinate time instant $5\ January\ 1987_{det}$ (which means *the whole day of*). However, the fact that somebody is leaving for Paris on January 5th, 1987 can be represented as an indeterminate time instant $5\ January\ 1987_{indet}$ (which means *some time on that day*). Hence, each discrete time instant is either *determinate* or *indeterminate*, corresponding to the two different interpretations. 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 determinate continuous interval $[5\ January\ 1987_{cont}, 6\ January\ 1987_{cont})$.

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.

Time spans Discrete and continuous determinate spans represent complete information about a duration of time. A discrete determinate span is a summation of distinct granularities with integer coefficients e.g., $5\ days$ or $2\ months\ +\ 5\ days$. Similarly, a continuous determinate span is a summation of distinct granularities with real coefficients e.g., $0.31\ hours$ or $5.2\ minutes\ +\ 0.15\ seconds$.

Discrete and continuous indeterminate spans represent incomplete information about a duration of time. They have lower and upper bounds that are determinate spans. For example, $1\ day\ \sim\ 2\ days$ is a discrete indeterminate span that can be interpreted as "a time period between one and two days."

The detailed inheritance hierarchy of a temporal structure is given in Figure 3 which shows the types and generic properties that are used to model various kinds of determinacy-domain-based temporal primitives.

Properties defined on time instants allow an instant to be compared with another instant; an instant to be subtracted from another instant to find the time duration between the two; and a time span to be added to or sub-

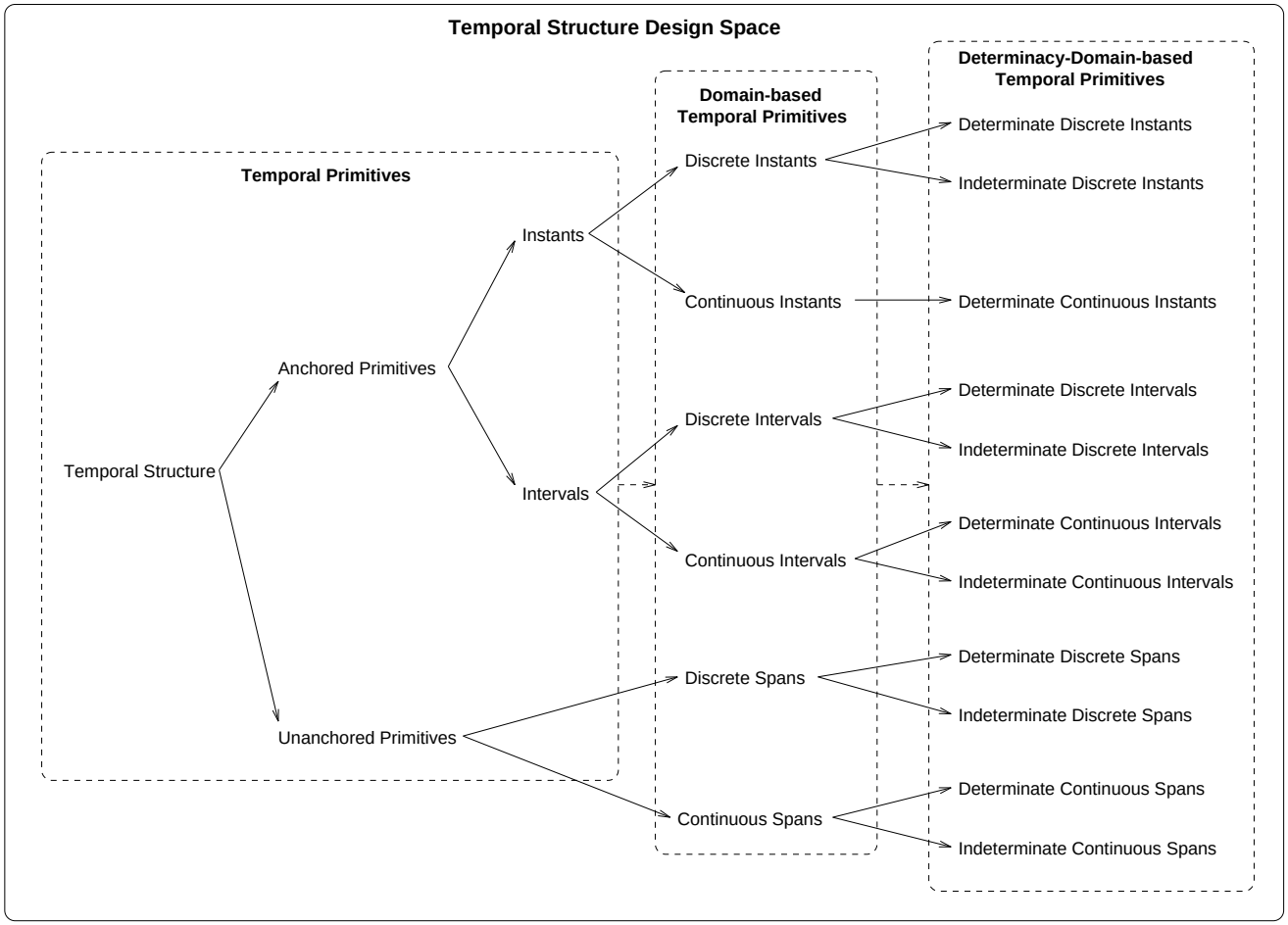

Figure 2: Design Space of a Temporal Structure

tracted from an instant to return another instant. Furthermore, properties *P_calendar* and *P_calElements* are used to link time instants to calendars which serve as a representational scheme for temporal primitives (see Section 2.1.2). *P_calendar* returns the calendar which the instant belongs to and *P_calElements* returns a list of the calendric elements in a time instant. For example *P_calendar* applied to the time instant 15 June 1995 would return *Gregorian*, while the application of *P_calElements* to the same time instant would return $(1995, June, 15)$.

Similarly, properties defined on time intervals include unary operations which return the lower bound, upper bound and length of the interval; ordering operations which define Allen's interval algebra [2]; and set-theoretic operations.

Properties defined on time spans enable comparison and arithmetic operations between spans. Additionally, properties *P_coefficient* and *P_calGranularities* are used as representational properties and provide a link between time spans and calendars (see Section 2.1.2). *P_coefficient* returns the (real) coefficient of a time span given a specific calendric granularity. For example, $(5\ days)\cdot$ *P_coefficient*(*day*) returns 5.0. *P_calGranularities* returns a collection of calendric granularities in a time span. For example, the property application $(1\ month + 5\ days)\cdot$ *P_calGranularities* returns $\{day, month\}$.

We note that (see Figure 3) the properties *P_succ* and *P_pred* are defined in all the types involving discrete primitives. This problem can be eliminated by refactoring the concerned types and using multiple inheritance. More specifically, an abstract type called `T_discrete` can be introduced, and the properties *P_succ* and and *P_pred* defined on it. All the types involving discrete primitives can then be made subtypes of `T_discrete`. A similar approach can be used to factor the types that define properties *P_lb* and *P_ub*. An abstract type called `T_bounds` can be introduced. with the properties *P_lb* and *P_ub* defined on it. The `T_interval` type and the types involving indetermi-

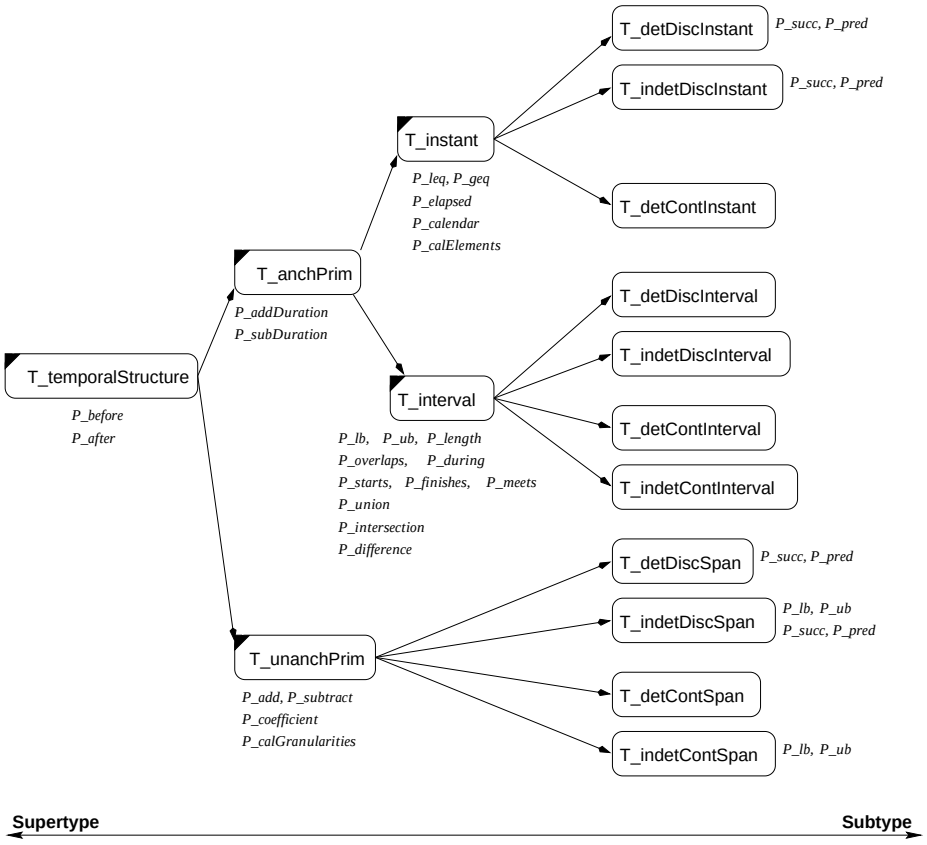

Figure 3: The Inheritance Hierarchy of a Temporal Structure

nate spans can then be made subtypes of `T_bounds`. Alternatively, the concept of multiple subtyping hierarchies can be used to collect semantically related types together and avoid the duplication of properties [9]. For example, the unanchored primitives hierarchy can be re-structured as shown in Figure 4.

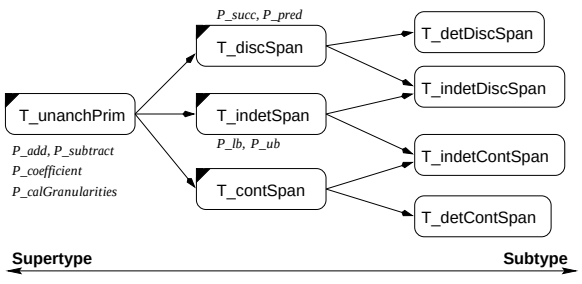

Figure 4: Multiple Subtyping Hierarchy for Unanchored Temporal Primitives

2.1.2 Temporal Representation

For human readability, it is important to have a representational scheme in which the temporal primitives can be made human readable and usable. This is achieved by means of calendars. A calendar is composed of an origin, a set of calendric granularities, and a set of conversion functions. The origin marks the start of a calendar. Calendric granularities define the reasonable time units (e.g., *minute*, *day*, *month*) that can be used in conjunction with this calendar to represent temporal primitives. A calendric granularity also has a list of *calendric elements*. For example in the Gregorian calendar, the calendric granularity *day* has the calendric elements *Sunday*, *Monday*, . . . , *Saturday*. Similarly in the Academic calendar, the calendric granularity *semester* has the calendric elements *Fall*, *Winter*, *Spring*, and *Summer*. The conversion functions establish the conversion rules between calendric granularities of a calendar.

Since all calendars have the same structure, a single type, called `T_calendar` can be used to model different calendars, where instances represent different calendars. The basic properties of a calendar are, *P_origin*,

P_calGranularities, and *P_functions*. These allow each calendar to define its origin, calendric granularities, and the conversion functions between different calendric granularities.

Example 2.1 Figure 5 shows four instances of `T_calendar` – the **Gregorian**, **Lunar**, **Academic**, and **Fiscal** calendars. The origin of the Gregorian calendar is given as the span $1582\ years$ from the start of time since it was proclaimed in 1582 by Pope Gregory XIII as a reform of the Julian calendar. The calendric granularities in the Gregorian calendar are the standard ones, **year**, **month**, **day**, etc. The origin of the Academic calendar shown in Figure 5 is assumed to be the span $1908\ academicYears$ having started in the year 1908, which is the establishment date of the University of Alberta. The Academic calendar has similar calendric granularities as the Gregorian calendar and defines a new calendric granularity of **semester**. The semantics of the Lunar and Fiscal calendars could similarly be defined. □

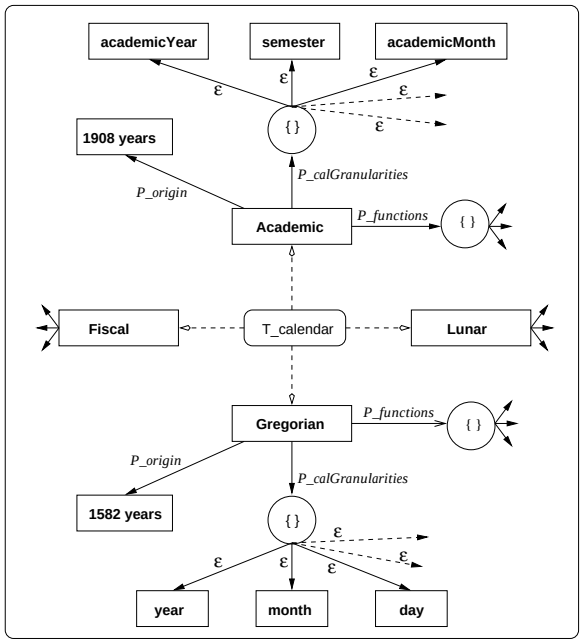

Figure 5: Temporal Representational Examples

2.1.3 Temporal Order

We now have the means of designing the temporal structure and the temporal representation of a temporal model. The next step is to provide an ordering scheme for the temporal primitives. This constitutes the third building block of our design space.

A temporal order can be classified as being *linear* or *branching*. In a linear order, time flows from past to future in an ordered manner. In a branching order, time is linear in the past up to a certain point, when it branches out into alternate futures. 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).

The different types of temporal orders are dependent on each other. A *sub-linear* order is one in which the temporal primitives (time intervals) are allowed to overlap, while a *linear* order is one in which the temporal primitives (time intervals) are not allowed to overlap. Every linear order is also a sub-linear order. A branching order is essentially made up of sub-linear orders. The relationship between temporal orders is shown in Figure 6.

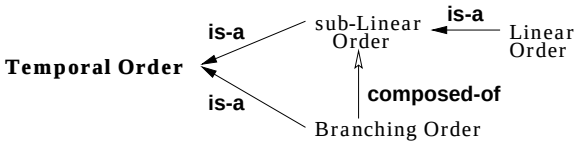

Figure 6: Temporal Order Relationships

The hierarchy in Figure 7 gives the various types and properties which model different temporal orders.

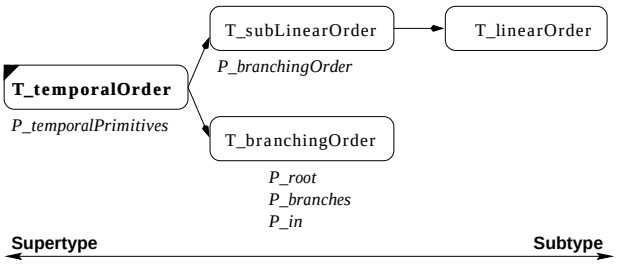

Figure 7: The Hierarchy of Temporal Orders

2.1.4 Temporal History

So far we have considered the various features of time; its structure, the way it is represented, and how it is ordered. The final building block of the design space of temporal models makes it possible to associate time with entities to model different temporal histories.

One requirement of a temporal model is an ability to associate temporal primitives with real-world entities. More

specifically, the temporal model should adequately represent and manage real-world entities as they evolve over time. An entity assumes different values over time. The set of these values forms the *temporal history* of the entity.

Two basic types of temporal histories are considered in databases which incorporate time. These are *valid* and *transaction* time histories [20]. Valid time denotes the time when an entity becomes effective (begins to model reality), while transaction time represents the time when a transaction is posted to the database. Usually valid and transaction times are the same. Other temporal histories include *event* time [16], and *user-defined* time histories. Unlike valid and transaction times the semantics of user-defined time is provided by the user, and not supported by the database management system. Since valid, transaction, event, and user-defined histories can have different semantics, they are orthogonal.

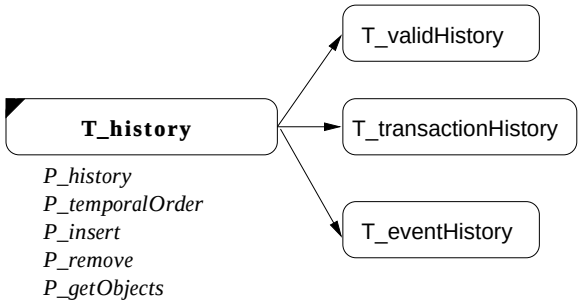

Figure 8: The Types and Properties for Temporal Histories

Figure 8 shows the various types that could be used to model these different histories. A temporal history consists of objects and their associated timestamps.

Property *P_history* defined in `T_history` returns a collection of all *timestamped* objects that comprise the history. A history object also knows the temporal order of its temporal primitives. The property *P_temporalOrder* returns the temporal order (which is an object of type `T_temporalOrder`) associated with a history object. The temporal order basically orders the time intervals (or time instants) in the history. Another property defined on history objects, *P_insert*, timestamps and inserts an object in the history. The *P_validObjects* property allows the user to get the objects in the history that were valid at (during) a given temporal primitive.

2.2 Relationships between Design Dimensions

In the previous section we described the building blocks (design dimensions) for temporal models and identified the design space of each dimension. We now look at the interactions between the design dimensions. This will enable us to put the building blocks together and structure the design space for temporal models.

A temporal history is composed of entities which are ordered in time. This temporal ordering is over a collection of temporal primitives in the history, which in turn are represented in a certain manner. Hence, the four dimensions can be linked via a "has-a" relationship as shown in Figure 9.

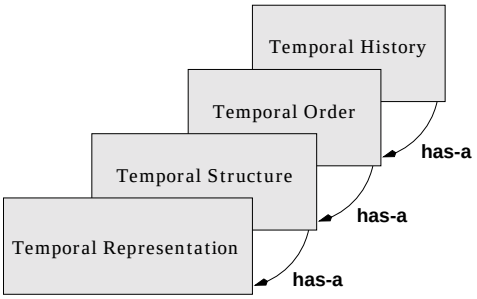

Figure 9: Relationships between the Temporal Design Dimensions

Basically, a temporal model can be envisioned as having a notion of time, which has an underlying temporal structure, a means to represent the temporal structure, and different temporal orders to order the temporal primitives within a temporal structure. This notion of time, when combined with certain entities could be used to represent certain temporal histories in the temporal model.

A temporal model can support one or more of valid, transaction, event, and user-defined histories. Each history in turn has a certain temporal order. This temporal order has properties which are defined by the type of temporal history (linear or branching). A linear history may or may not allow overlapping of anchored temporal primitives that belong to it. If it does not allow overlapping, then such a history defines a total order on the anchored temporal primitives that belong to it. Otherwise, it defines a partial order on its anchored temporal primitives. Each order can then have a temporal structure which can comprise of all or a subset of the 11 different temporal primitives. Finally, different calendars can be defined as a means to represent the temporal primitives.

The four dimensions are modeled by the respective types shown in Figure 10. The "has a" relationship between the dimensions is modeled using appropriate properties and is represented in the figure by dashed arrows between the respective types. An object of `T_temporalHistory` represents a temporal history. Its temporal order is obtained using the *P_temporalOrder* property. A temporal order is an object of type `T_temporalOrder` and has a certain temporal structure which is obtained using the *P_temporalPrimitives* property. The temporal structure is an object of type `T_temporalStructure`. The property *P_calendar*

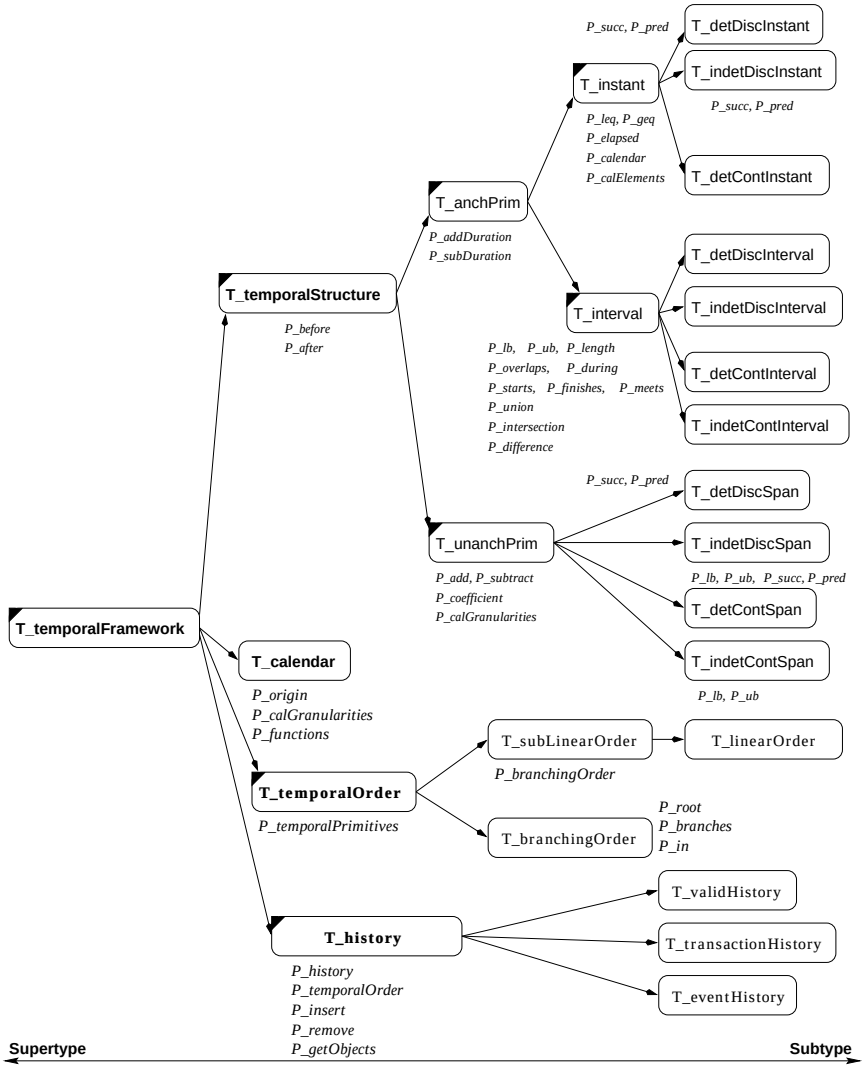

Figure 11: The Inheritance Hierarchy for the Temporal Framework

gives the instance of `T_calendar` which is used to represent the temporal structure.

The relationships shown in Figure 10 provide us with a temporal framework which encompasses the design space for temporal models. The detailed inheritance hierarchy, shown in Figure 11, is based on the design dimensions identified in Section 2 and their various features which are given in Figures 3, 7, and 8.

As we described in Section 2.1.1, refactoring of types and multiple inheritance can be used to handle identical properties that are defined over different types in the inheritance hierarchy shown in Figure 11. The framework can now be tailored for the temporal needs of different applications and temporal models. This is illustrated in Section 3.

3 Tailoring the Temporal Framework

In this section, we illustrate how the temporal framework that is defined in Section 2 can be tailored to accommodate applications and temporal models which have different temporal requirements. In the first two sub-sections, we give examples of two real-world applications that have different temporal needs. In the last sub-section, we give an example of a temporal object model and show how the model can be derived from the temporal framework.

3.1 Clinical Data Management

In this section we give a real-world example from clinical data management that illustrates the four design dimensions and the relationships between them which were discussed in Section 2.

During the course of a patient's illness, different blood

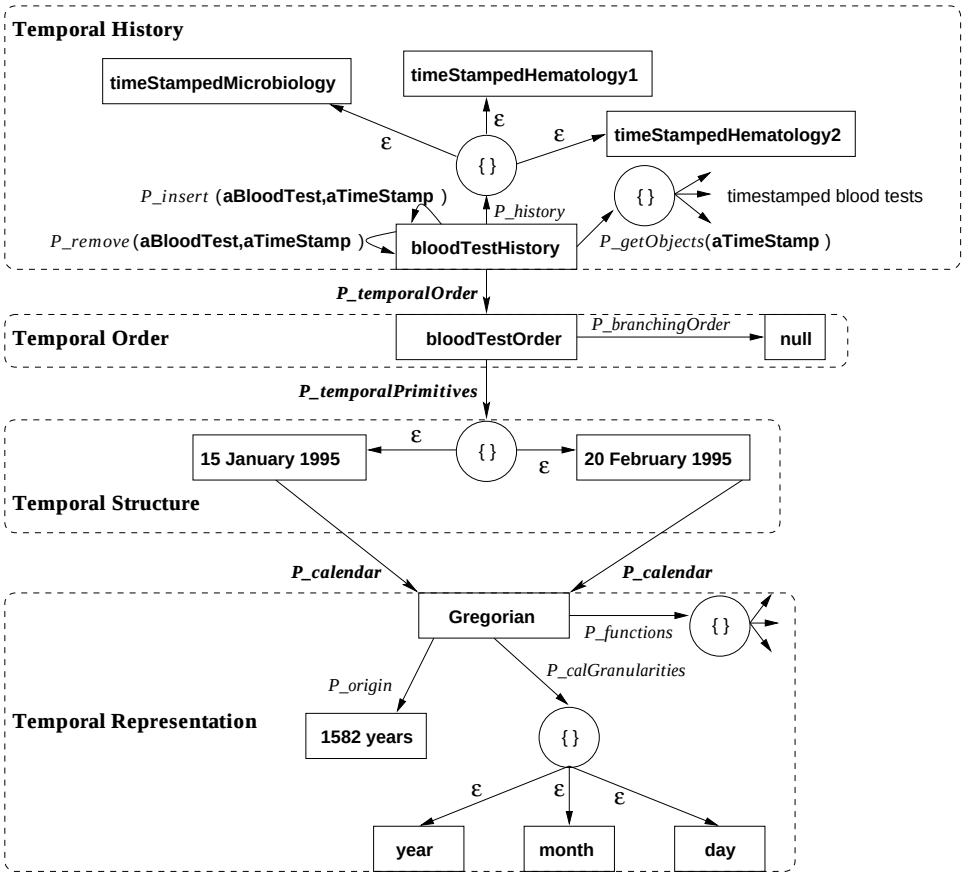

Figure 12: A Patient's Blood Test History

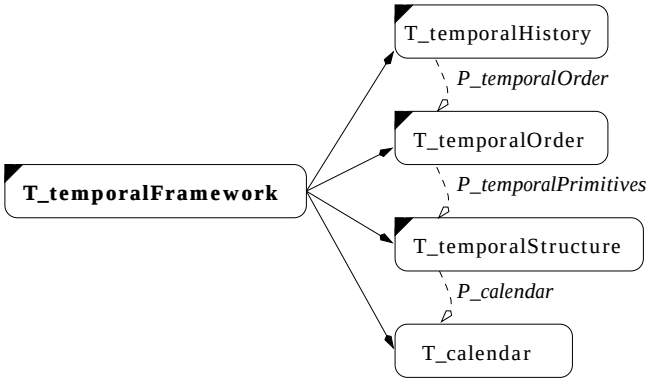

Figure 10: Relationships between Design Dimensions Types

tests are administered. It is usually the case that multiple blood tests of the patient are carried out on the same day. Suppose the patient was suspected of having an infection of the blood, and therefore, had two different blood tests on 15 January 1995. These were the diagnostic hematology and microbiology blood tests. 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 awaited. A repeat hematology test was ordered on 20 February 1995. Suppose each blood test is represented by an object of the type `T_bloodTest`. The valid history of the patient's blood tests can then be represented in the object database as an object of type `T_validHistory`. Let us call this object bloodTestHistory. To record the hematology and microbiology blood tests, the objects hematology and microbiology whose type is `T_bloodTest` are first created and then entered into the object database using the following property applications:

$$\text{bloodTestHistory}.P\_insert(\text{microbiology}, 15\ January\ 1995)$$
$$\text{bloodTestHistory}.P\_insert(\text{hematology1}, 15\ January\ 1995)$$
$$\text{bloodTestHistory}.P\_insert(\text{hematology2}, 20\ February\ 1995)$$

If subsequently there is a need to determine which blood tests the patient took in January 1995, this would be accomplished by the property application $\text{bloodTestHistory}.P\_getObjects([1\ \ January\ \ 1995,$

$31\ January\ 1995]$). This would return a collection of timestamped objects of T_bloodTest representing all the blood tests the patient took in January 1995. These objects would be the (timestamped) hematology1 and the (timestamped) microbiology.

Figure 12 shows the different temporal features that are needed to keep track of a patient's blood tests over the course of a particular illness. The figure also illustrates the relationships between the different design dimensions of the temporal framework.

The patient has a blood test history represented by the object bloodTestHistory. The *P_history* property when applied to bloodTestHistory results in a collection object whose members are the timestamped objects timeStampedMicrobiology, timeStampedHematology1, and timeStampedHematology2. The *P_insert*(bloodTestHistory) function object updates the blood test history when given an object of type T_bloodTest and an anchored temporal primitive. Similarly, the *P_getObjects*(bloodTestHistory) function object returns a collection of timestamped objects when given an anchored temporal primitive.

Applying the property *P_temporalOrder* to bloodTestHistory results in the object bloodTestOrder which represents the temporal order on different blood tests in bloodTestHistory. bloodTestOrder has a certain temporal structure which is obtained by applying the *P_temporalPrimitives* property. Finally, the primitives in the temporal structure are represented using the Gregorian calendar, Gregorian and the calendric granularities year, month, and day.

Let us now consider the various temporal features required to represent the different blood tests taken by a patient. Anchored, discrete, and determinate temporal primitives are required to model the dates on which the patient takes different blood tests. These dates are represented using the Gregorian calendar. Since the blood tests take place on specific days, the temporal primitives during which the patient took blood tests form a total order. Lastly, a valid time history is used to keep track of the different times the blood tests were carried out. To support these temporal features, the temporal framework can be reconfigured with the appropriate types and properties. These are given in Figure 13.

3.2 Time Series Management

The management of time series is important in many application areas such as finance, banking, and economic research. One of the main features of time series management is extensive calendar support [6, 13]. Calendars map time points to their corresponding data and provide a platform for granularity conversions and temporal queries. Therefore, the temporal requirements of a time series management system include elaborate calendric functionality (which allows the definition of multiple calendars and granularities) and variable temporal structure (which supports both anchored and unanchored temporal primitives, and the different operations on them).

Figure 14 shows how the temporal requirements of a time series management system can be modeled using the types and properties of the temporal framework. We note from the figure that only the temporal structure and temporal representation design dimensions are used to represent the temporal needs of a time series. This demonstrates that it is not necessary for an application requiring temporal features to have all four design dimensions in order to be accommodated in the framework. One or more of the design dimensions specified in Section 2.1 can be used as long as the design criteria shown in Figures 9 holds.

3.3 TOODM - A Temporal Object-Oriented Data Model

In this section, we illustrate how the temporal framework can accommodate the temporal features of different temporal object models. Due to space limitations, we concentrate on Rose & Segev's temporal object-oriented data model (TOODM) [16] since it uses object types and inheritance to model temporality. We refer the reader to [8] for details on how the temporal framework also accommodates the temporal features of the rest of the temporal object models [24, 11, 5, 15, 7, 3] that have appeared in the literature.

3.3.1 Overview of Temporal Features

TOODM was designed by extending an object-oriented entity-relationship data model to incorporate temporal structures and constraints. The functionality of TOODM includes: specification and enforcement of temporal constraints; support for past, present, and future time; support for different type and instance histories; and allowance for retro/proactive updates. The type hierarchy of the TOODM system defined types used to model temporality is given in Figure 15. The boxes with a dashed border represent types that have been introduced to model time, while the rest of the boxes represent basic types.

The Object type is the root of the type tree. The type V-Class is used to represent user-defined versionable classes. More specifically, if the instance variables, messages/methods, or constraints of a type are allowed to change (maintain histories), the type must be defined as a subtype of V-Class.

The Ptypes type models primitive types and is used to represent objects which do not have any instance variables. Ptypes usually serve as domains for the instance

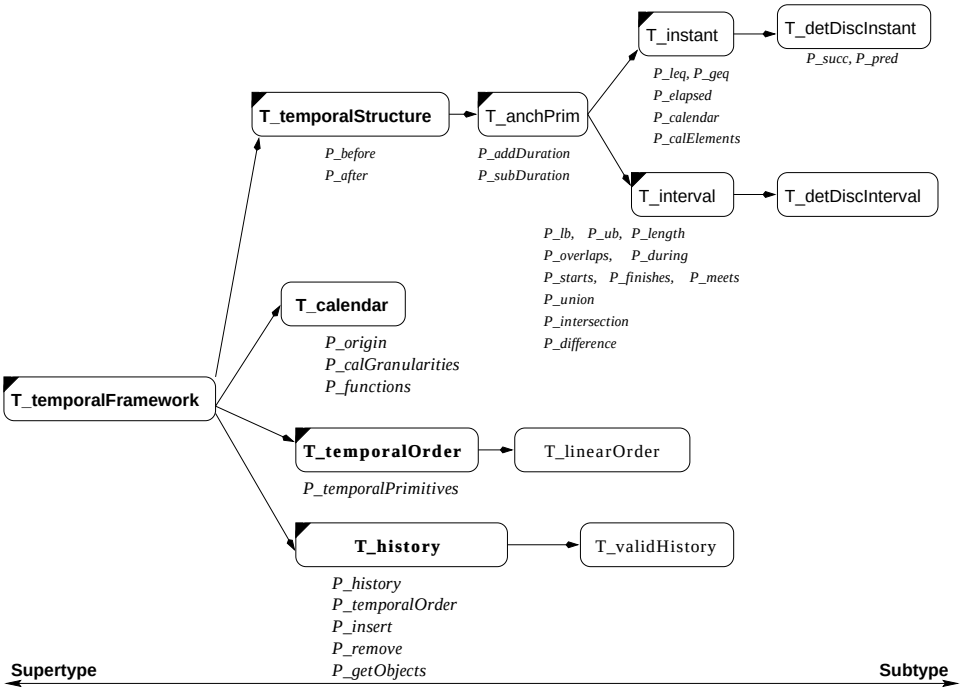

Figure 13: The Temporal Framework Inheritance Hierarchy for the Clinical Applicati on

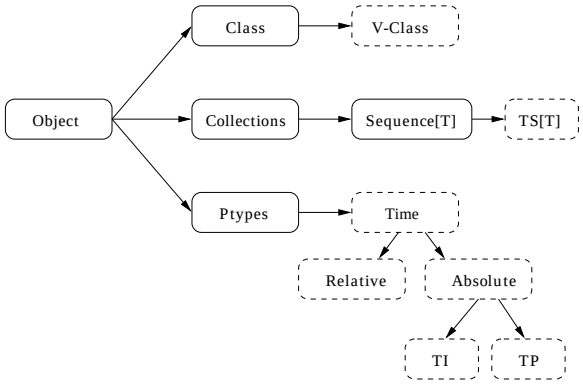

Figure 15: System Defined Temporal Types in TOODM

variables of other objects. The `Time` primitive type is used to represent temporal primitives. The `TP` type represents time points, while the `TI` type represents time intervals. Time points can have specific different calendar granularities, namely *Year*, *Month*, *Day*, *Week*, *Hour*, *Minute*, and *Second*.

The `TS[T]` type represents a time sequence which is a collection of objects ordered on time. `TS[T]` is a parametric type with the type `T` representing a user or system defined type upon which a time sequence is being defined. For every time-varying attribute in a (versionable) class, a corresponding subclass (of `TS[T]`) is defined to represent the time sequence (history) of that attribute. For example, if the salary history of an employee is to be maintained, a subclass (e.g., `TS[Salary]`) of `TS[T]` has to be defined so that the salary instance variable in the employee class (which is defined as a subclass of `V-Class`) can refer to it to obtain the salary history of a particular employee. The history of an object of type `TS[T]` is represented as a pair <T,TL>, where T is the data type and TL defines the different timelines and their granularities that are associated with T. Three timelines are allowed in TOODM: valid time, record (transaction) time, and event time (the time an event occurred). Each timeline associated with an object is comprised of time points or time intervals and has an underlying granularity.

3.3.2 Representing the Temporal Features of TOODM in the Temporal Framework

TOODM supports both anchored and unanchored primitives. These are modeled by the `Absolute` and `Relative` types shown in Figure 15. The anchored temporal primitives supported are time instants and time intervals. A continuous time domain is used to perceive the temporal primitives. Finally, the temporal primitives are determinate.

Time points and time intervals are represented by using the Gregorian calendar with granularities *Year*, *Month*, *Day*, *Week*, *Hour*, *Minute*, and *Second*. Translations be-

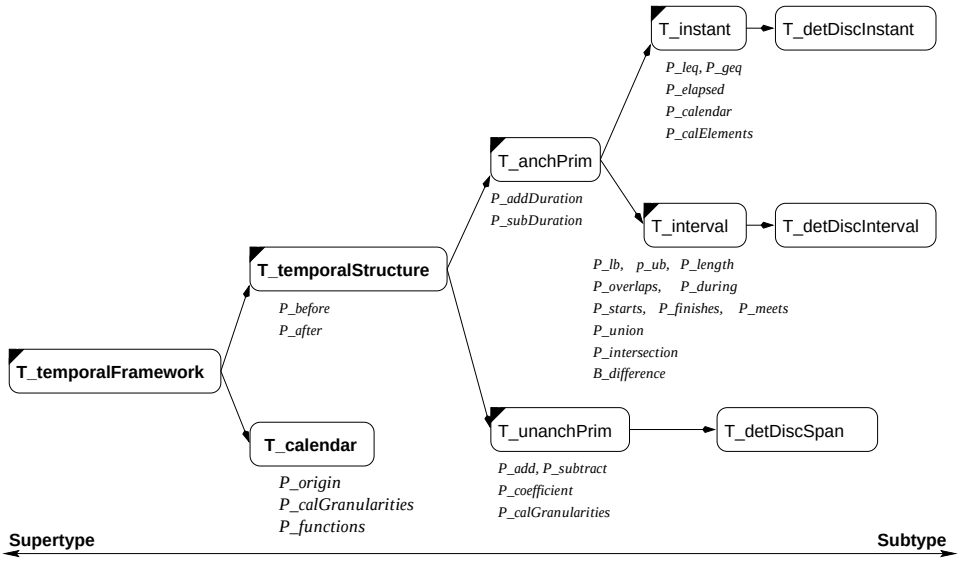

Figure 14: The Temporal Framework Inheritance Hierarchy for Time Series Management

Structure			Representation	Order	History
Primitives	Domain	Determinacy			
Anchored	Continuous	Determinate	Gregorian Calendar	Total Linear	Valid
Unanchored					Transaction
					Event

Table 1: Temporal Design Dimension Features of TOODM

tween granularities in operations are provided, with the default being to convert to the coarser granularity. A (presumably total) linear order of time is used to order the primitives in a temporal sequence. TOODM combines time with facts to model different temporal histories, namely, valid, transaction, and event time histories. Table 1 summarizes the temporal features (design space) of TOODM according to the design dimensions for temporal models that were described in Section 2.1. Figure 16 shows the type system instance of our temporal framework that corresponds to the TOODM time types shown in Figure 15 and described in Table 1.

The `Time` primitive type is represented using the `T_temporalStructure` type. The `TP` and `TI` types are represented using the `T_instant` and `T_interval` types, respectively. Similarly, the `Relative` type is represented using the `T_unanchPrim` type. Since TOODM supports continuous and determinate temporal primitives, the (concrete) types `T_detContInstant`, `T_detContInterval`, and `T_detContSpan` are used to model continuous and determinate instants, intervals, and spans, respectively.

The Gregorian calendar and its different calendric granularities are modeled using the `T_calendar` type. Time points and time intervals are ordered using the `T_linearOrder` type. Time sequences represented by the `TS[T]` type are modeled by the history types in the temporal framework. More specifically, valid time (vt), record time (rt), and event time (et) are modeled using the `T_validHistory`, `T_transactionHistory`, and `T_eventHistory` types.

TOODM models valid, transaction and event histories all together in one structure as shown by the `TS[Salary]` type in the previous section. Our temporal framework, however, provides different types to model valid, transaction, and event histories to allow their respective semantics to be modeled. Moreover, it uses properties to access the various components of histories. For example, to represent the valid history of an employee's salary an object of type `T_validHistory` is first created. The *P_insert* property then inserts objects of type `T_integer` (representing salary values) and objects of type `T_interval` (representing time intervals) into the salary valid history object. The transaction and event time histories of the salary are similarly represented, except in these histories the *P_insert* property inserts timestamps which are time instants (i.e., objects of type `T_instant`).

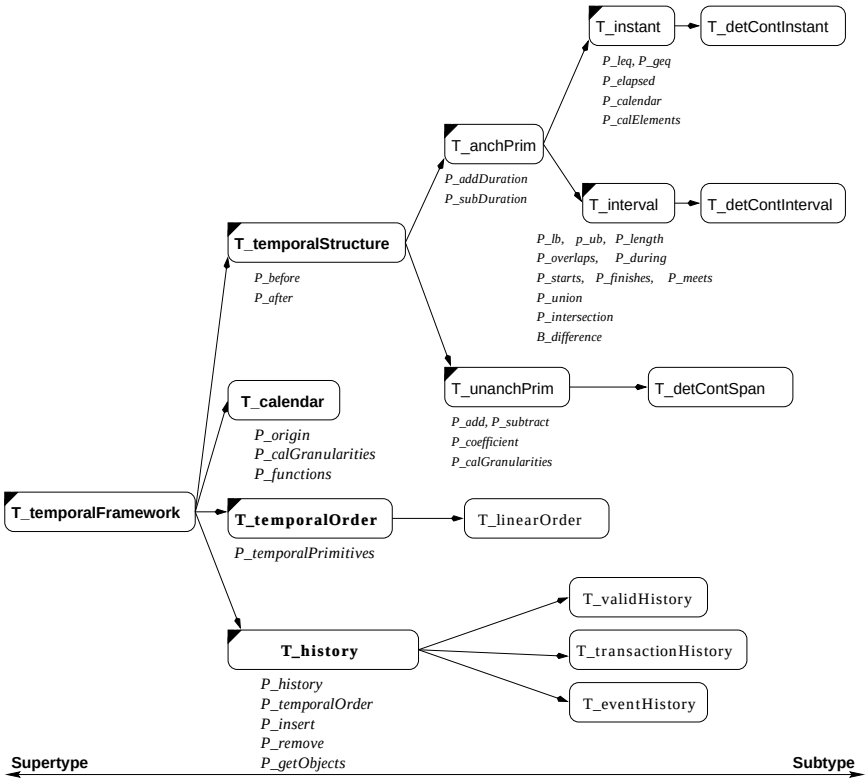

Figure 16: The Temporal Framework Inheritance Hierarchy for TOODM

4 Discussion and Conclusions

In this paper the different design dimensions that span the design space of temporal object models were identified. Object-oriented techniques were then used in designing an infrastructure which supports the diverse notions of time under a single framework. We also demonstrated the expressiveness of the framework by showing how it can be used to accommodate the temporal needs of different real-world applications, and also reflect different temporal object models that have been reported in the literature.

This temporal framework subsumes the work of Wuu & Dayal [26] in that it provides the user or database designer with explicit types and properties to model the diverse features of time. Wuu & Dayal provide an abstract *time* type to model the most general semantics of time which can then be subtyped (by the user or database designer) to model the various notions of time required by specific applications. However, their approach requires significant support from the user, including specification of the temporal schema. We contend that specifying a schema for modeling time is complex, and certainly not trivial. It is therefore imperative for temporal object models to have a temporal infrastructure from which users can choose the temporal features they need.

Using the object-oriented inheritance hierarchy to structure the design space of temporal object models and identify the dependencies within and among the design dimensions helped us simplify the presentation of the otherwise complex domain of time. The focus in this work has been on the unified provision of temporal features which can be used by temporal object models according to their temporal needs. Once these are in place, the model can then define other object-oriented features to support its application domain.

The diverse features of the temporal domain are also identified in [19]. The focus however, is on comparing various temporal object models and query languages based on their ability to support valid and transaction time histories. In this paper we show how the generic aspects of temporal models can be captured and described in a single framework. In [14] a temporal reference framework for multimedia synchronization is proposed and used to compare existing temporal specification schemes and their relationships to multimedia synchronization. The focus however, is on different forms of temporal specification, and not on different notions of time. The model of time used concentrates only on temporal primitives and their representation schemes.

The temporal framework has been implemented in C++. Furthermore, a toolkit has been developed in Perl/Tk in order to allow users/temporal model designers to interact with the framework at a high level and generate specific framework instances for their own applications. The temporal framework also gives a means to compare temporal objects models according to the design dimensions that were identified in Section 2. This will help identify the strengths and weaknesses of the different temporal objects models. A research direction worth pursuing would be to compare the temporal framework with frameworks for time representation developed in the field of artificial intelligence (for example, [17]).

References

[1] A-R. Adl-Tabatabai, T. Gross, and G-Y. Lueh. Code Reuse in an Optimizing Compiler. In *Proc. of the Int'l Conf on Object-Oriented Programming: Systems, Languages, and Applications - OOPSLA '96*, pages 51–68, October 1996.

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

[3] E. Bertino, E. Ferrari, and G. Guerrini. A Formal Temporal Object-Oriented Data Model. In *Proc. 5th Int'l Conf. on Extending Database Technology*, March 1996.

[4] R.H. Campbell, V.F. Russo, and G.M. Johnston. The Design of a Multiprocessor Operating System. In *Proceedings of the USENIX 1987 C++ Conference*, 1987.

[5] 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.

[6] W. Dreyer, A.K. Dittrich, and D. Schmidt. An Object-Oriented Data Model for a Time Series Management System. In *Proc. 7th International Working Conference on Scientific and Statistical Database Management*, September 1994.

[7] I.A. Goralwalla and M.T. Özsu. Temporal Extensions to a Uniform Behavioral Object Model. In *Proc. 12th Int'l Conf. on the Entity Relationship Approach (ER'93)*, pages 115–127, December 1993.

[8] I.A. Goralwalla, M.T. Özsu, and D. Szafron. An Object-Oriented Framework for Temporal Data Models. In O. Etzion, S. Jajodia, and S. Sripada, editors, *Temporal Databases: Research and Practice*. Springer Verlag, 1997. To appear.

[9] W.H. Harrison, H. Kilov, H.L. Ossher, and I. Simmonds. From Dynamic Supertypes to Subjects: a Natural way to Specify and Develop Systems. *IBM Systems Journal*, 35(2):244–256, 1996.

[10] R.E. Johnson and B. Foote. Designing Reusable Classes. *Journal of Object-Oriented Programming*, 1(2):22–35, 1988.

[11] 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, June 1992.

[12] N. Kline. An Update of the Temporal Database Bibliography. *ACM SIGMOD Record*, 22(4):66–80, December 1993.

[13] J.Y. Lee, R. Elmasri, and J. Won. Specification of Calendars and Time Series for Temporal Databases. In *Proc. 15th International Conference on Conceptual Modeling (ER'96)*, October 1996. Proceedings published as Lecture Notes in Computer Science, Volume 1157, Bernhard Thalheim (editor), Springer-Verlag, 1996.

[14] M.J. Perez-Luque and T.D.C. Little. A Temporal Reference Framework for Multimedia Synchronization. *IEEE Journal on Selected Areas in Communications*, 14(1):36–51, January 1996.

[15] N. Pissinou and K. Makki. A Framework for Temporal Object Databases. In *Proc. First Int'l. Conf. on Information and Knowledge Management*, November 1992.

[16] 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.

[17] Y. Shahar. A Framework for Knowledge-Based Temporal Abstraction. *Artificial Intelligence*, 90(1-2):79–133, 1997.

[18] R. Snodgrass. Research Concerning Time in Databases: Project Summaries. *ACM SIGMOD Record*, 15(4), December 1986.

[19] R. Snodgrass. Temporal Object-Oriented Databases: A Critical Comparison. In W. Kim, editor, *Modern Database Systems: The Object Model, Interoperability and Beyond*, pages 386–408. Addison-Wesley/ACM Press, 1995.

[20] 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.

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

[22] M.D. Soo. Bibliography on Temporal Databases. *ACM SIGMOD Record*, 20(1):14–23, 1991.

[23] R. Stam and R. Snodgrass. A Bibliography on Temporal Databases1. *IEEE Database Engineering*, 7(4):231–239, December 1988.

[24] S.Y.W. Su and H.M. Chen. A Temporal Knowledge Representation Model OSAM*/T and its Query Language OQL/T. In *Proc. 17th Int'l Conf. on Very Large Data bases*, 1991.

[25] V.J. Tsotras and A. Kumar. Temporal Database Bibliography Update. *ACM SIGMOD Record*, 25(1), March 1996.

[26] 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, Tempe, USA, February 1992.