An Object-Oriented Framework for Temporal Data Models

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. Therefore, the first step of this work is a design space for temporal models which accommodates multiple notions of time, thereby classifying design alternatives for temporal models. The design space is then represented by exploiting object-oriented features to model the different aspects of time. 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 allow the various notions of time to be reflected in a single framework. 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. Furthermore, it can be used to compare and analyze different temporal object models with respect to the design dimensions.

1 Introduction

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.

To accommodate the temporal needs of different applications, there has been extensive research activity on temporal data models in the last decade [Sno86,SS88,Soo91,Kli93,TK96]. Most of this research has concentrated on the definition of a particular temporal model and its incorporation into a (relational or object-oriented) database management system (DBMS).

To appear in *Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (editors), Springer Verlag, 1998.*

The early research on temporal data models 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 that only support a discrete and linear model of time.

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, [RS91,SC91,WD92,KS92,CITB92,BFG97]). These models differ in the functionality that they offer, however as in relational systems, they assume a set of fixed notions of time. Wuu & Dayal [WD92] 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 a few support a *branching* model. 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 [Sno87], respectively.

These commonalities suggest a need for combining the diverse features of time under a single infrastructure that is extensible and allows design reuse. In this paper, we present an object-oriented framework [JF88] 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.

The objectives of this work are fourfold. The first objective is to identify the design dimensions that span the design space for temporal models. This will classify design alternatives for temporal models. The design space is then represented by exploiting object-oriented features to model the different aspects of time. The second objective is to show how the temporal framework can be tailored to accommodate real-world applications that have different temporal needs. The third objective is to show how the various existing temporal object models can be represented within this framework. The final objective is to use the framework to analyze and compare the different temporal object models based on the design dimensions. In particular, the [RS91,SC91,KS92,PM92,CITB92,BFG97] temporal object models are considered. The work of Wuu & Dayal [WD92] and Cheng & Gadia [CG93] (which follows a similar methodology as [WD92]) are not considered since they do not provide concrete notions of time in their models. Object models which support versioning using time [KGBW90,WLH90,SRH90,Sci94] usually follow a structural embedding of temporality within type definitions. Thus, the notion of temporal objects is lost since the model knows nothing about temporality. Moreover, most temporal version models use the `Date` function call which is provided by the system. For example, though the EXTRA-V version model [Sci94] supports "valid" and "transaction" time, it does so by timestamping attributes using system provided dates. This is limited in scope as no semantics of the various notions of time are provided. Since these models are not "temporal object models" in the strict sense of the term, we do not include them in this study.

We can draw a parallel between our work and similar (albeit on a much larger scale) approaches used in *Choices* [CJR87] and *cmcc* [ATGL96]. *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 presented in this paper 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 five 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. In Section 4 object-oriented techniques are used to compare and analyze different temporal object models with respect to the design dimensions. Section 5 sum-

marizes the work presented in this paper, discusses related work, 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 (abstractions of methods and attributes in traditional object-oriented terminology). 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* – provides the underlying ontology and domains for time.
2. *Temporal Representation* – provides a means to represent time so that it is human readable.
3. *Temporal Order* – gives an ordering to time.
4. *Temporal History* – allows events and activities to be associated with time.

There are two parts to the description of a design dimension. First, we define a set of temporal features that the design dimension encompasses. Second, we explore relationships between the temporal features and describe the resulting design space for the design dimension. The design space consists of an architectural overview of abstract and concrete types corresponding to the temporal features, and a design overview which describes some of the key properties (operations) defined in the interface of the types. We do not describe the properties in detail since many of these are traditional temporal operations that have already appeared in the literature on temporal databases.

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, 8, 9, and 15 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.

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. We now give an overview of the features of a temporal structure and then identify the relationships that exist between them.

Components

1. **Temporal Primitives**
   Temporal primitives can either be *anchored* (*absolute*) or *unanchored* (*relative*) [Sno92]. For example, 31 *July* 1995 is an anchored temporal primitive since we know exactly where it is located on the time axis, whereas 31 *days* is an unanchored temporal primitive since it can stand for any block of 31 consecutive days on the time axis.
   There is only one unanchored primitive, called the *span*. A span is a duration of time with a known length, but no specific starting and ending anchor points. There are two anchored primitives, the *instant* (*moment*, *chronon*) and the *interval*. An instant is a specific anchored moment in time, e.g., 31 *July* 1995. An interval is a duration of time between two specific anchor points (instants) which are the lower and upper bounds of the interval, e.g., [15 *June* 1995, 31 *July* 1995].
2. **Temporal Domain**
   The temporal domain of a temporal structure defines a scale for the temporal primitives. A temporal domain can be *continuous* or *discrete*. Discrete domains map temporal primitives to the set of integers. That is, for any temporal primitive in a discrete time domain, there is a unique successor and predecessor. Continuous domains map temporal primitives to the set of real numbers. Between any two temporal primitives of a continuous time domain, another temporal primitive exists. Most of the research in the context of temporal databases has assumed that the temporal domain is discrete. Several arguments in favor of using a discrete temporal domain are made by Snodgrass [Sno92] including the imprecision of clocking instruments, compatibility with natural language

references, possibility of modeling events which have duration, and practicality of implementing a continuous temporal data model. However, Chomicki [Cho94] argues that the continuous (dense) temporal domain is very useful in mathematics and physics. Furthermore, continuous time provides a useful abstraction if time is thought of as discrete but with instants that are very close. In this case, the set of time instants may be very large which in turn may be difficult to implement efficiently. Chomicki further argues that query evaluation in the context of constraint databases [KKR90,Rev90] has been shown to be easier in continuous domains than in discrete domains. Continuous temporal domains have also been used to facilitate full abstract semantics in reasoning about concurrent programs [BKP86].

3. **Temporal Determinacy**
   There are many real world cases where we have complete knowledge of the time or the duration of a particular activity. For example, the time interval allowed for students to complete their *Introduction to Database Management Systems* examination is known for certain. This is an example of a determinate temporal primitive. However, there are cases when the knowledge of the time or the duration of a particular activity is known only to a certain extent. For example, we do not know the exact time instant when the Earth was formed though we may speculate on an approximate time for this event. In this case, the temporal primitive is indeterminate. Indeterminate temporal information is also prevalent in various sources such as granularity, dating techniques, future planning, and unknown or imprecise event times [DS93]. Since the ultimate purpose of a temporal model is to represent real temporal information, it is desirable for such a model to be able to capture both determinate and indeterminate temporal primitives.

Design Space

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 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

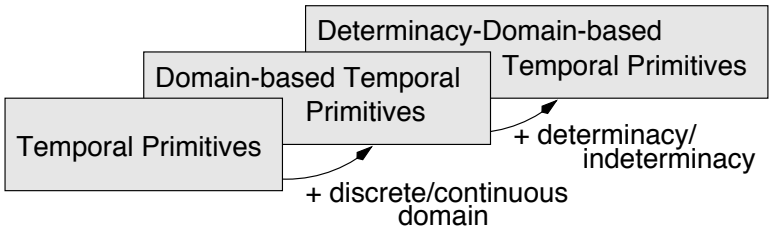

Fig. 1. Building a Temporal Structure

indeterminate. The difference between them is that a continuous determinate interval denotes that the activity associated with it occurs during the *whole* interval, while a continuous indeterminate interval denotes that the activity associated with it occurs *sometime* during the interval. Continuous intervals have 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 5 January 1997. This fact can be expressed using a determinate time instant 5 $January\ 1997_{det}$ (which means *the whole day of*). However, the fact that somebody is leaving for Paris on 5 January 1997 can be represented as an indeterminate time instant 5 $January\ 1997_{indet}$ (which means *some time on that day*). Hence, each discrete time instant is either *determinate* or *indeterminate*, corresponding to the two different interpretations. Discrete time instants are analogous to continuous time intervals. Every determinate (indeterminate) discrete time instant has a granularity (G_i) associated with it. This granularity determines the mapping of the given determinate (indeterminate) discrete time instant I_{det} (I_{indet}) to the domain of continuous time instants. The mapping is defined as follows:

$$I_{det} \mapsto [I_{cont}, I_{cont} + G_i)$$
$$I_{indet} \mapsto [I_{cont} \sim I_{cont} + G_i)$$

Here I_{cont} denotes the counterpart of I_{det} and I_{indet} in the domain of continuous time instants. This is exemplified by the mapping of the discrete determinate instant 5 $January\ 1997_{det}$ to the continuous determinate interval [5 $January\ 1997_{cont}$, 6 $January\ 1997_{cont}$). In this case $G_i = G_{days} = 1\ day$. A formal treatment of the different types of instants and mappings is given in [GLÖS97].

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.,

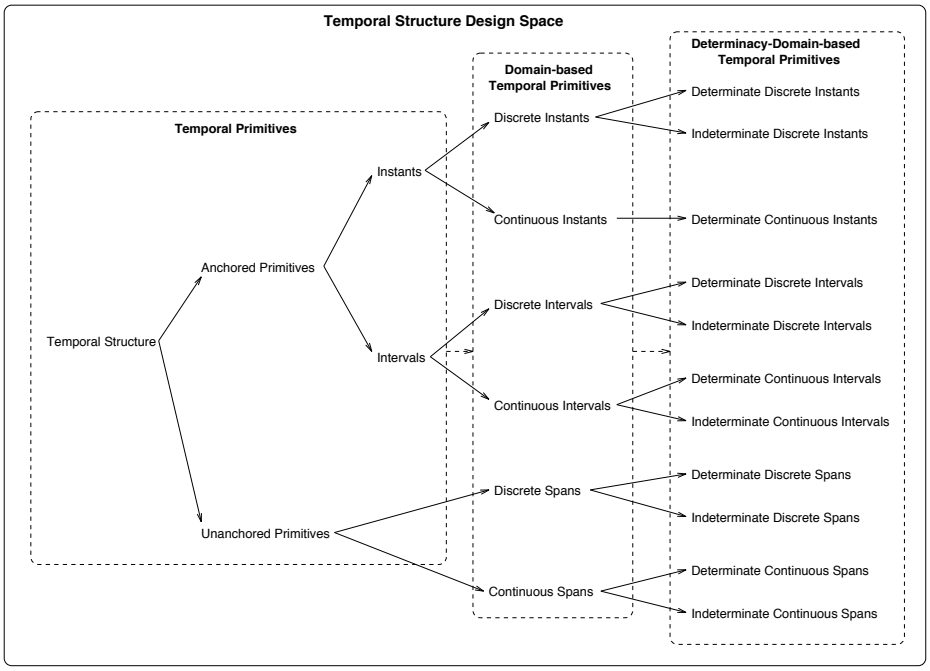

Fig. 2. Design Space of a Temporal Structure

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 mapping of the temporal structure to an object type hierarchy 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 subtracted 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). *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

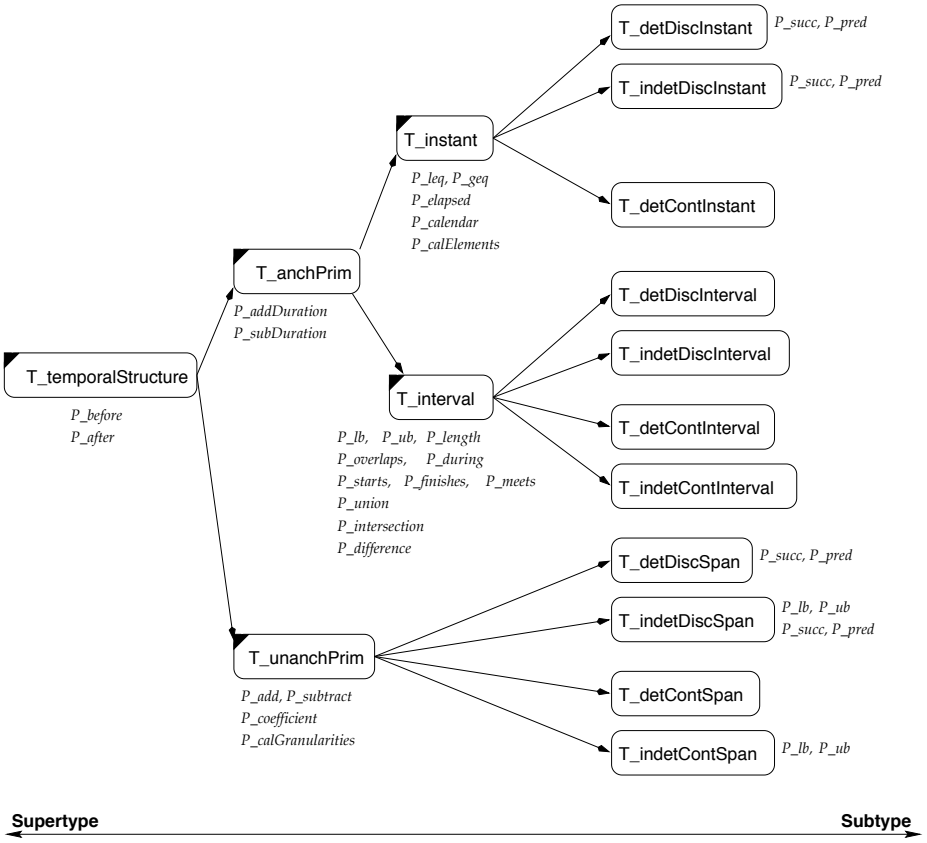

Fig. 3. The Inheritance Hierarchy of a Temporal Structure

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 [All84]; and set-theoretic operations.

Properties defined on time spans enable comparison and arithmetic operations between spans. The *P_before* and *P_after* properties are refined for time spans to model the semantics of $<$ and $>$, respectively. 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). *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 both discrete instant and span primitives. This redundancy 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 indeterminate 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 [HKOS96]. For example, the unanchored primitives hierarchy can be re-structured as shown in Figure 4.

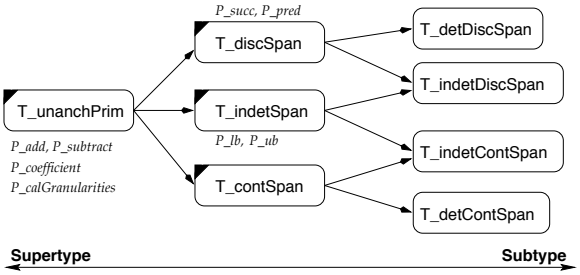

Fig. 4. Multiple Subtyping Hierarchy for Unanchored Temporal Primitives

Temporal Representation

Components. 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. Common calendars include the *Gregorian* and *Lunar* calendars. Educational institutions also use *Academic* calendars.

Calendars are comprised of different time units of varying granularities that enable the representation of different temporal primitives. In many applications, it is desirable to have multiple calendars that have different calendric granularities. For example, in financial trading, multiple calendars with different time units and operations need to be available to capture the semantics of financial data [CS93,CSS94]. In time series management, extensive calendar support is also required [DDS94,LEW96].

A calendar should be able to support multiple granularities since temporal information processed by a DBMS is usually available in multiple granularities. Such information is prevalent in various sources. For example:

- *clinical data* – Physicians usually specify temporal clinical information for patients with varying granularities [CPP95,CPP96]. For example, "the patient suffered from abdominal pain for 2 hours and 20 minutes on June 15, 1996," "in 1990, the patient took a calcium antagonist for 3 months," "in October 1993, the patient had a second heart seizure."
- *real-time systems* – A process is usually composed of sub-processes that evolve according to times that have different granularities [CMR91]. For example, the temporal evolution of the basin in a hydroelectric plant depends on different sub-processes: the flow of water is measured daily; the opening and closing of radial gates is monitored every minute; and the electronic control has a granularity of microseconds.
- *geographic information systems* – Geographic information is usually specified according to a varying time scale [Flo91]. For example, vegetation fluctuates according to a seasonal cycle, while temperature varies daily.
- *office information systems* – temporal information is available in different time units of the Gregorian calendar [BP85,CR88,MPB92]. For example, employee wages are usually recorded in the time unit of hours while the history of sales are categorized according to months.

Design Space. 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[1]. 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 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

[1] We note that our definition of a calendar is different from that defined in [CS93,CSS94,LEW96] where structured collections of time intervals are termed as "calendars." Our definition adheres closely to the human understanding of a calendar. However, the extensibility feature of the framework allows any other notions of calendars to be incoporated easily under the temporal representation design dimension.

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.

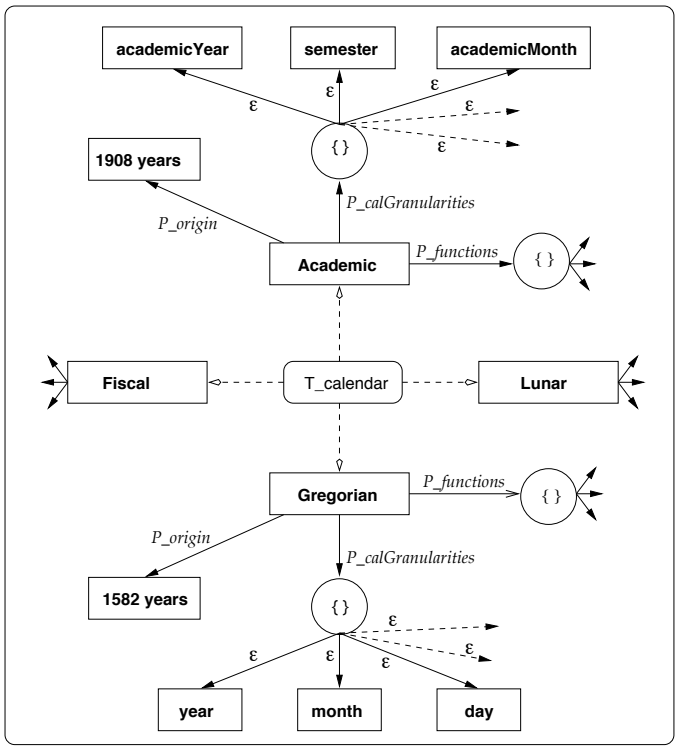

Fig. 5. Temporal Representational Examples

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.

Components. 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 linear model is used in applications such as office information systems. 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).

Design Space. 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.

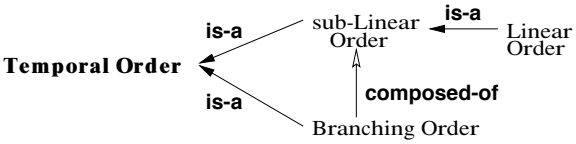

Fig. 6. Temporal Order Relationships

The hierarchy in Figure 7 gives the various types and properties which model different temporal orders[2].

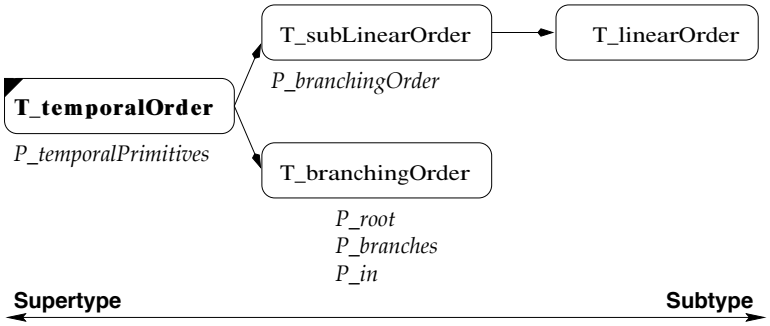

Fig. 7. The Hierarchy of Temporal Orders

[2] We do not consider *periodic* temporal orders in this work. These can easily be incorporated as a subtype of `T_temporalOrder`.

Example 2. Consider the operations that take place in a hospital on any particular day. It is usually the case that at any given time multiple operations are taking place. Let us assume an eye cataract surgery took place between 8am and 10am, a brain tumor surgery took place between 9am and 12pm, and an open heart surgery took place between 7am and 2pm on a certain day. Figure 8 shows a pictorial representation of operationsOrder, which is an object of type `T_subLinearOrder`. operationsOrder consists of the time intervals [08:00,10:00], [09:00,12:00], [07:00,14:00], and does not belong to any branching timeline. As seen in the figure, operationsOrder consists of intervals (representing the time periods during which the different surgeries took place) that overlap each other. Hence, operationsOrder is an example of a sub-linear order.

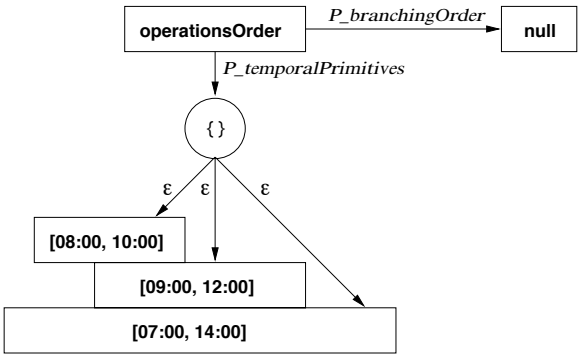

Fig. 8. An Example of a Sub-Linear Order.

Example 3. To illustrate the use of objects of type `T_linearOrder` which are total linear temporal orders, consider a patient with multiple pathologies, for example as a result of diabetes. The patient has to attend several special clinics, each on a different day. Hence, it follows that since the patient cannot attend more than one special clinic on any day, the temporal order of the patient's special clinics visit history is linear and totally ordered. Suppose the patient visited the opthalmology clinic on 10 January 1995, the cardiology clinic on 12 January 1995, and the neurology clinic on 3 February 1995. Figure 9 shows a pictorial representation of specialClinicOrder, which is an object of type `T_linearOrder`. As seen in the figure, specialClinicOrder is totally ordered as its time intervals do not overlap.

Example 4. Consider an observational pharmacoeconomic analysis of the changing trends, over a period of time, in the treatment of a chronic illness such as asthma [GÖS97]. The analysis would be performed using information gathered over a time period. At a fixed point during this period new guidelines for the

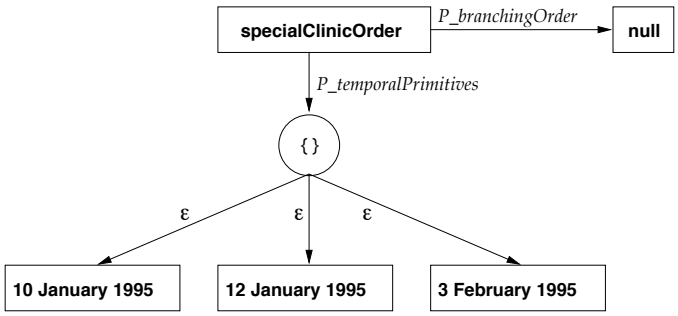

Fig. 9. An Example of a Linear Order.

treatment of asthma were released. At that point the population of patients known to have asthma are divided into those whose doctors continue the old established treatment, and those whose doctors, in accordance with new recommendations, change their treatment. Thus, the patients are divided into two groups, each group undergoing a different treatment for the same illness. The costs and benefits accrued over the trial period for each treatment are calculated. Since such a study consists of several alternative treatments to an illness, a branching timeline is the natural choice for modeling the timeline of the study. The point of branching is the time when the new guidelines for the treatment of the illness are implemented. Figure 10 shows the branching timeline for such a medical trial history.

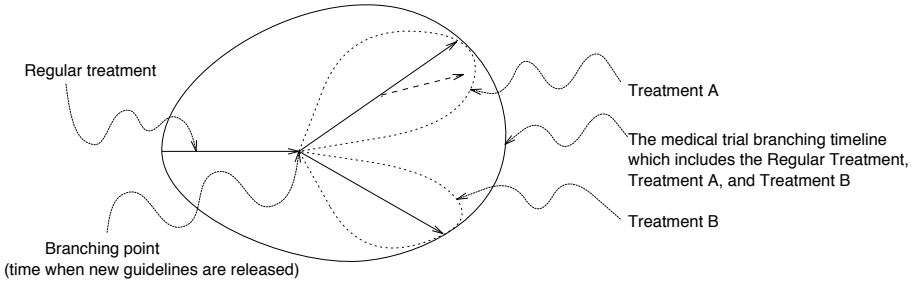

Fig. 10. An Example of a Branching Order.

The same branching timeline could as easily handle the situation where different versions of a particular treatment, say Treatment A, are implemented based on certain parameters. In this case, the "Treatment A" branch would in turn branch at a certain point into different Treatment A versions. This situation is also depicted in Figure 10.

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.

Components. One requirement of a temporal model is an ability to represent and manage real-world entities as they evolve over time and assume different states (values). 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 [SA85]. Valid time denotes the time when an entity is effective (models 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 [RS91,CK94] and *decision* time [EGS93] histories. Event (decision) time denotes the time the event occured in the real-world. Valid, transaction, and event times have been shown to be adequate in modeling temporal histories [CK94].

Design Space. Since valid, transaction, and event time histories have different semantics, they are orthogonal. Figure 11 shows the various types that could be used to model these different histories. A temporal history consists of objects and their associated timestamps.

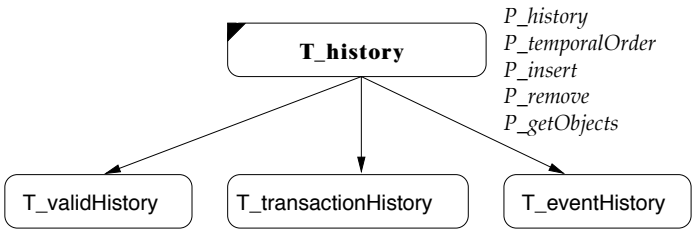

Fig. 11. The Types and Properties for Temporal Histories

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. Property *P_remove* drops a given object from the history at a specified temporal primitive. The *P_getObjects* property allows the user to get the objects in the history at (during) a given temporal primitive. The properties defined on `T_history` are refined in `T_validHistory`, `T_transactionHistory`, and

`T_eventHistory` types to model the semantics of the different kinds of histories. Moreover, each history type can define additional properties, if necessary, to model its particular semantics. The clinical example described in Section 3.1 illustrates the use of the properties defined on `T_history`.

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 the "has-a" relationship shown in Figure 12.

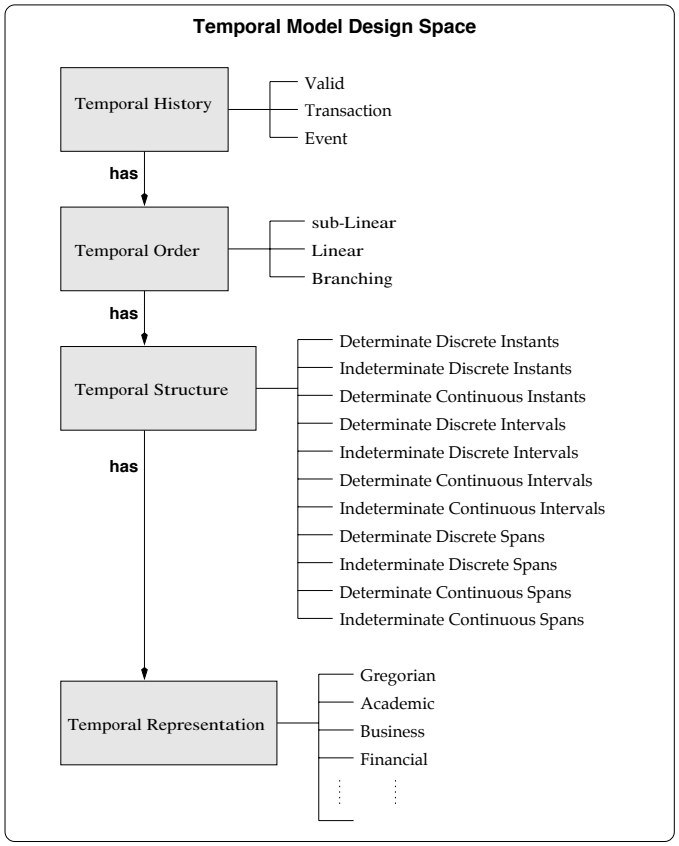

Fig. 12. Design Space for Temporal Models

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 application objects can be used to represent various temporal histories of the objects in the temporal model.

Figure 12 gives the design space for temporal models. 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 is comprised of all or a subset of the 11 different temporal primitives that are shown in Figure 2. Finally, different calendars can be defined as a means to represent the temporal primitives.

The four dimensions are modeled in an object system by the respective types shown in Figure 13. The "has a" relationship between the dimensions is modeled using properties as shown in the figure. 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* gives the instance of `T_calendar` which is used to represent the temporal structure.

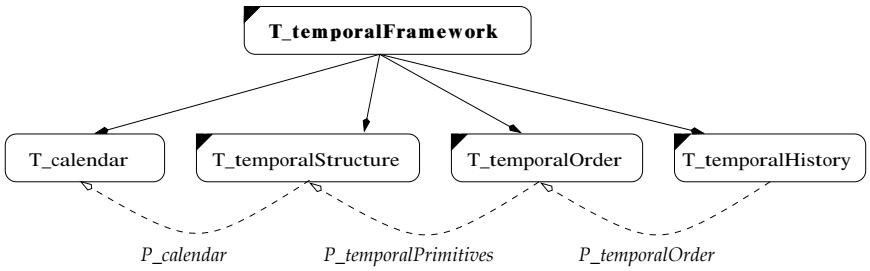

Fig. 13. Relationships between Design Dimensions Types

The relationships shown in Figure 13 provide a temporal framework which encompasses the design space for temporal models. The detailed type system, shown in Figure 14, is based on the design dimensions identified in Section 2 and their various features which are given in Figures 3, 7, and 11. As described in Section 2.1, 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 14. The framework can now be tailored for the temporal needs of different applications and temporal models. This is illustrated in Section 3.

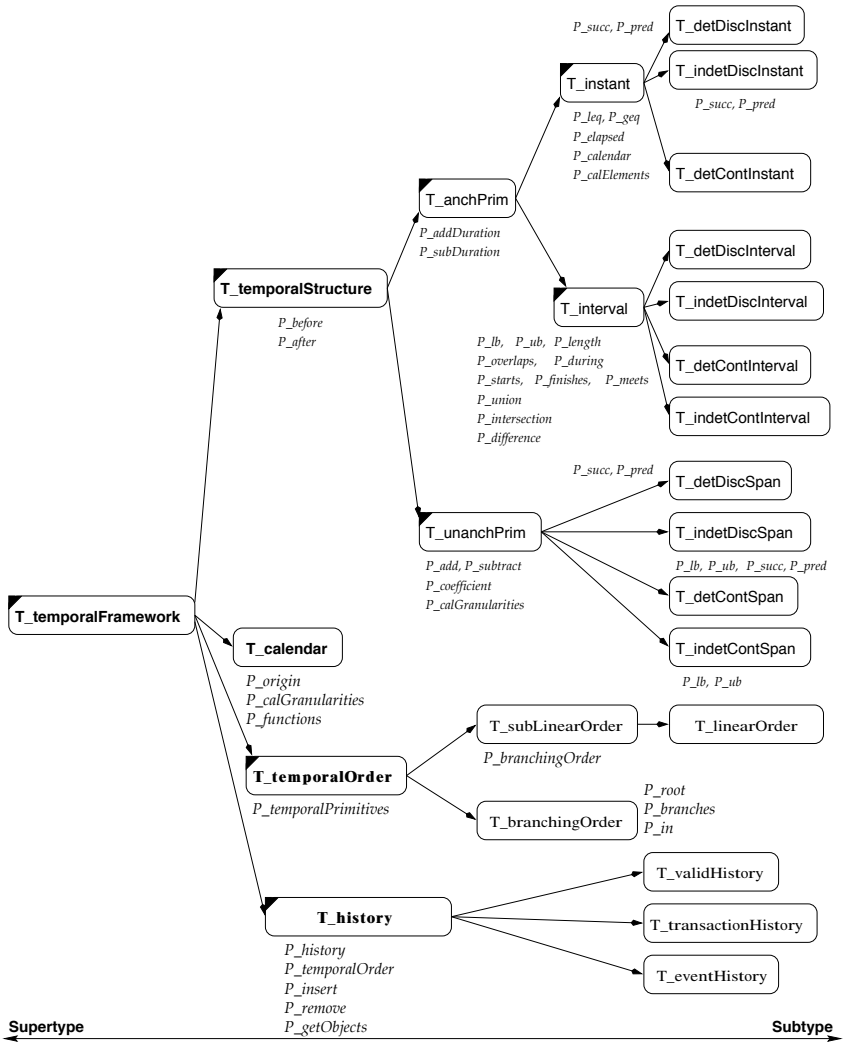

Fig. 14. The Inheritance Hierarchy for the Temporal Framework

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

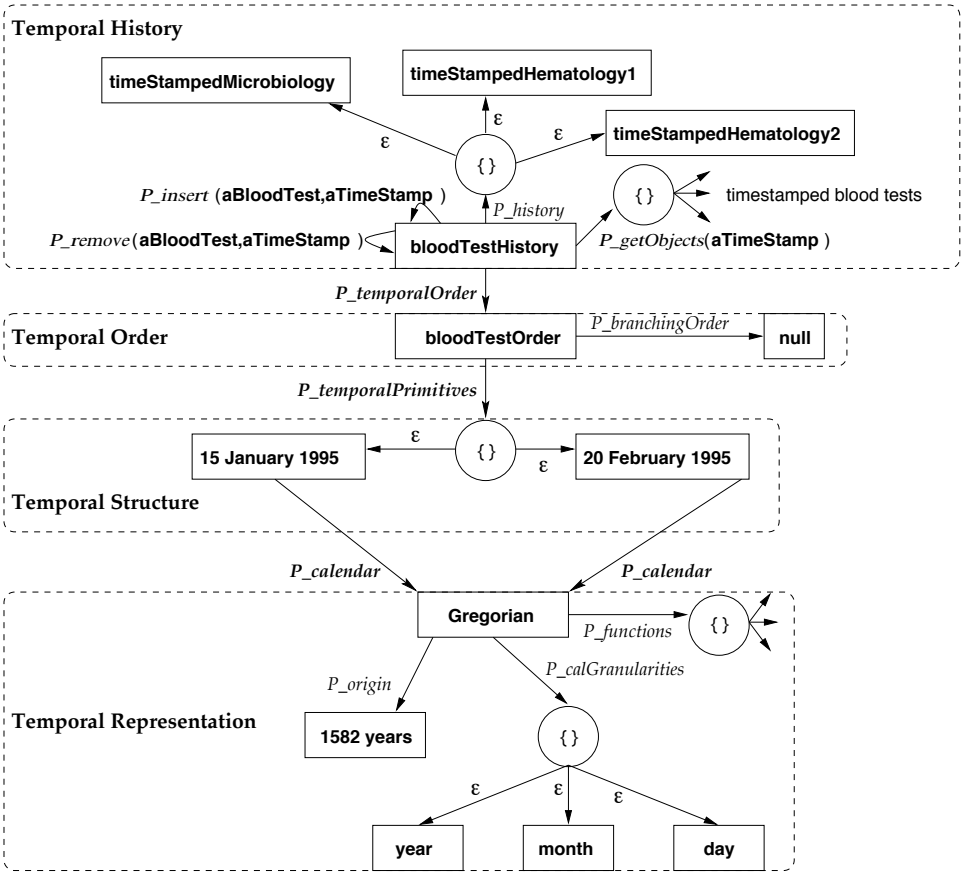

Fig. 15. A Patient's Blood Test History

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 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:

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

If subsequently there is a need to determine which blood tests the patient took in January 1995, this would be accomplished by the property application 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 15 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

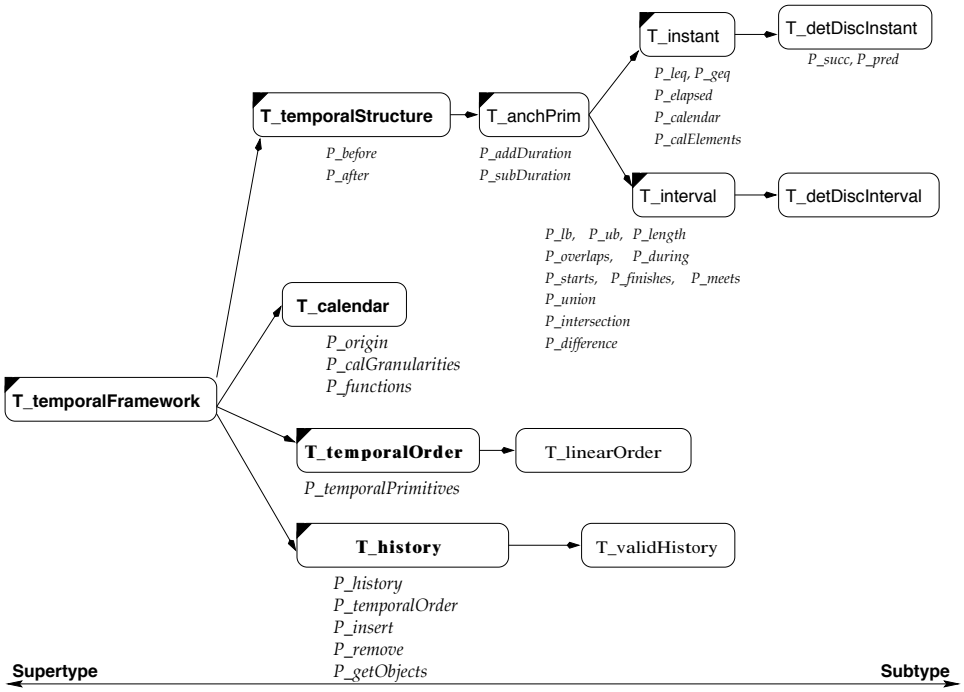

Fig. 16. The Temporal Framework Inheritance Hierarchy for the Clinical Applicati on

support these temporal features, the temporal framework can be reconfigured with the appropriate types and properties. These are given in Figure 16.

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 [DDS94,LEW96]. 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 17 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

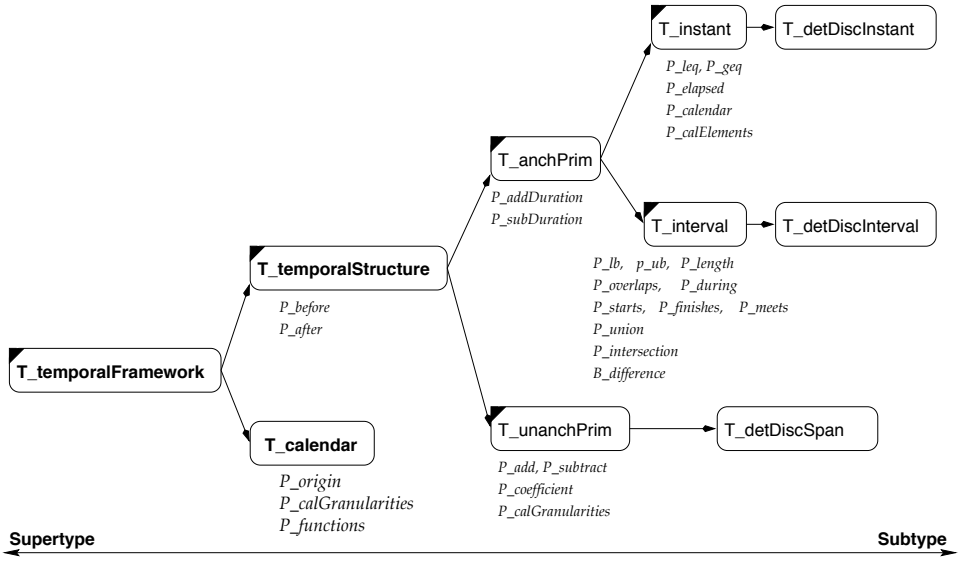

Fig. 17. The Temporal Framework Inheritance Hierarchy for Time Series Management

in Section 2.1 can be used as long as the design criteria shown in Figure 12 holds.

3.3 TOODM - A Temporal Object-Oriented Data Model

In this section, we identify the temporal features of Rose & Segev's temporal object-oriented data model (TOODM) [RS91] according to the design dimensions described in Section 2.1, and show how these can be accommodated in the temporal framework. We specifically concentrate on TOODM since it uses object types and inheritance to model temporality. The temporal features of the rest of the reported temporal object models [SC91,KS92,CITB92,PM92,BFG97] are summarized and compared in Section 4. We first give an overview of the temporal features of TOODM and then show how these features can be derived using the types and properties of our temporal framework. There is no doubt that TOODM has more functionality to offer in addition to temporality, but presenting that is beyond the scope of this work.

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 18. The boxes with a dashed border represent types

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

Table 1. Temporal Design Dimension Features of TOODM

that have been introduced to model time, while the rest of the boxes represent basic types.

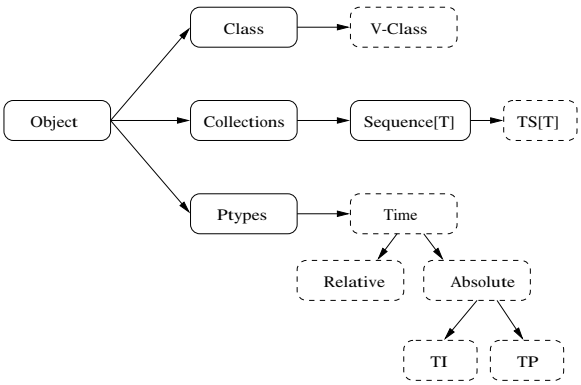

Fig. 18. System Defined Temporal Types in TOODM

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 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.

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 18. 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 between 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 19 shows the type system instance of our temporal framework that corresponds to the TOODM time types shown in Figure 18 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

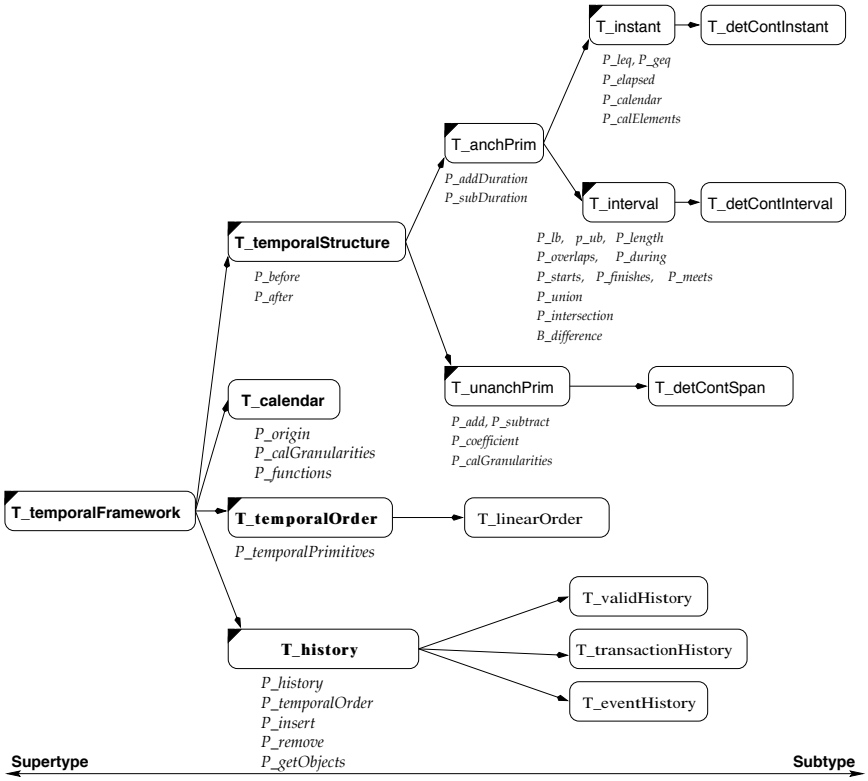

Fig. 19. The Temporal Framework Inheritance Hierarchy for TOODM

`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`).

4 Comparison of Temporal Object Models

In this section we use the temporal framework to compare and analyze the temporal object models [RS91,SC91,KS92,CITB92,PM92,BFG97] that have appeared in recent literature. The temporal features of these models are summarized in Tables 1 and 2. Our criteria in comparing different temporal object models is based on the design dimensions identified in Section 2.1. It is true that the models may have other (salient) temporal differences, but our concern in this work is comparing their temporal features in terms of the framework defined in Section 2.

Similar to the methodology used in Section 2, object-oriented techniques are used to classify temporal object models according to each design dimension. This gives us an indication of how temporal object models range in their provision for different temporal features of a design dimension – from the most powerful model (i.e., the one having the most number of temporal features) to the least powerful model (i.e., the one having the least number of temporal features).

Model	Structure			Representation	Order	History
	Primitives	Domain	Determinacy			
OSAM*/T	Anchored	Discrete	Determinate	N/A	Linear	Valid
TMAD	Anchored	Discrete	Determinate	Gregorian Calendar	Linear	Valid
						Transaction
TEDM	Anchored	Discrete	Determinate	N/A	Linear	Valid
						Transaction
						Event
T-3DIS	Anchored	Discrete	Determinate	Gregorian Calendar	Partial	Valid
T-Chimera	Anchored	Discrete	Determinate	N/A	Linear	Valid

Table 2. Design Dimension Features of different Temporal Object Models

Temporal Structure. It can be noticed from Tables 1 and 2 that most of the models support a very simple temporal structure, consisting of anchored primitives which are discrete and determinate. In fact, all models in Table 2 support the *same* temporal structure, which consists of discrete and determinate anchored temporal primitives. These primitives can be accommodated in the temporal framework by the `T_anchPrim`, `T_instant`, `T_detDiscinstant`, `T_interval`, and `T_detDiscInterval` types, and their respective properties. The temporal structure of TOODM is slightly enhanced with the presence of unanchored primitives. TOODM is also the only model that supports the continuous temporal domain.

Figure 20 shows how the type inheritance hierarchy is used to classify temporal object models according to their temporal structures. The temporal structures of OSAM*/T, TMAD, TEDM, T-3DIS, and T-Chimera can be modeled by a single type – that representing temporal primitives that are anchored, discrete, and determinate. This means that any of these models can be used to provide temporal support for applications that need a temporal structure comprised of anchored temporal primitives which are discrete and determinate. Similarly, the temporal structure of TOODM can be be modeled by a type which represents anchored and unanchored temporal primitives that are continuous and determinate. This implies that TOODM is the only model that can support applications requiring a continuous time domain, or unanchored temporal primitives.

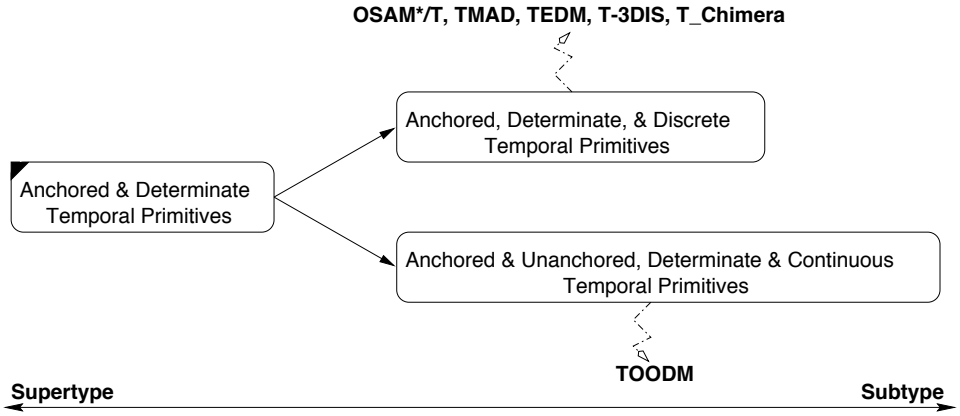

Fig. 20. Classification of Temporal Object Models according to their Temporal Structures

Temporal Representation. Temporal primitives in the OSAM*/T [SC91], TEDM [CITB92], and T-Chimera [BFG97] models are simply represented using natural numbers. The models do not provide any additional representational scheme which supports calendars and different granularities. The granularity of the temporal primitives is dependent on the application using the model. When a calendric representational scheme is provided for the temporal primitives, it is comprised of a single underlying calendar, which is usually Gregorian. This is the case in the TOODM [RS91], TMAD[KS92], and T-3DIS [PM92] models.

Temporal Order. All models shown in Tables 1 and 2, except T-3DIS, support a linear temporal order. The T-3DIS model supports a sub-linear temporal order. These temporal orders are accommodated in the temporal framework using the `T_subLinearOrder` and `T_linearOrder` types. Figure 21 shows how the models can be classified in an inheritance type hierarchy according to their temporal orders. The type modeling a partial linear order of time sits at the root of the hierarchy and represents the T-3DIS model. Since a total linear order is also a partial order, the models supporting total linear orders can be represented by a direct subtype of the root type.

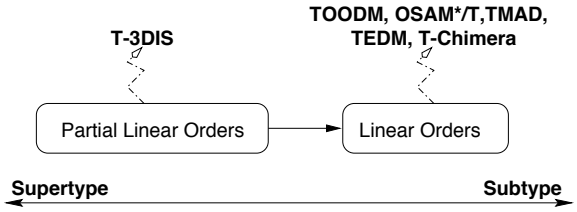

Fig. 21. Classification of Temporal Object Models according to their Temporal Orders

Temporal History. Tables 1 and 2 show how the temporal object models range in their support for the different types of temporal histories. Figure 22 shows how the models can be classified according to the temporal histories they support using a type inheritance hierarchy. The root type in Figure 22 represents the models which only support valid time histories. These are the OSAM*/T, T-3DIS, and T-Chimera models. A direct subtype of the root type inherits the valid time history and provides transaction time history as well. This type represents the TMAD model. Similarly, the rest of the subtypes inherit different histories from their supertypes and add new histories to their type as shown in Figure 22. From Figure 22, we see that applications requiring only valid time histories can be supported by all models; applications requiring valid and transaction time can be supported by the TMAD, TEDM, and TOODM models; and applications requiring valid, transaction, and event time can be supported by the TEDM and TOODM models.

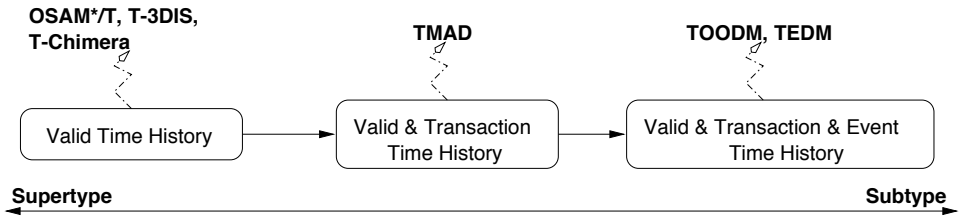

Fig. 22. Classification of Temporal Object Models according to their Temporal Histories

Overall Classification. Having classified the temporal object models according to the individual design dimensions, we now treat the models as points in the design space and use the object-oriented inheritance hierarchy to compare the models on all the temporal features of the design dimensions that they support. Figure 23 gives an inheritance hierarchy in which types are used to represent the different models, and the temporal features supported by the models are used as a criteria for inheritance.

The abstract type at the root of the hierarchy represents the least powerful temporal object model which supports a temporal structure comprised of anchored primitives which are discrete and determinate, no temporal representational scheme, a partial linear order, and a valid time history. This type has two immediate subtypes. The first subtype represents the OSAM*/T and the T-Chimera models. It inherits all the features of the root type and refines its partial linear order to a total linear order. Similarly, the second subtype represents the T-3DIS model, inherits all the features of the root type, and adds a representational scheme which supports the Gregorian calendar. The type representing OSAM*/T and T-Chimera also has two subtypes. The first subtype represents the TEDM model and has all the features of its supertype with the additional features of transaction and event time histories.

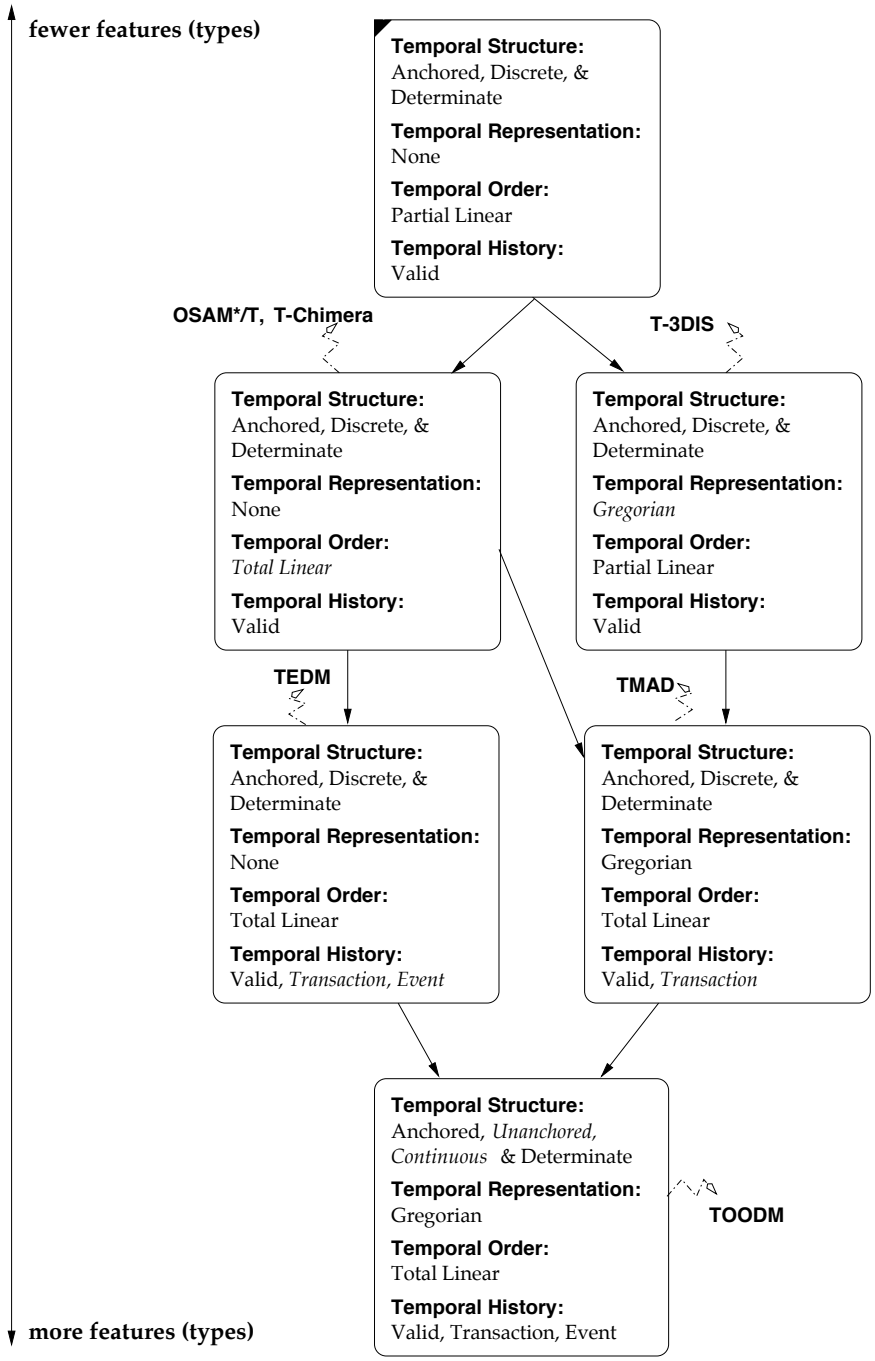

Fig. 23. Overall Classification of Temporal Object Models

The second subtype (which is also a subtype of the type representing T-3DIS from which it inherits the representational scheme) represents the TMAD model. This type has the additional feature of the transaction time history. A direct subtype of the types representing TEDM and TMAD represents the TOODM model. The type representing TOODM inherits the representational scheme from the type representing TMAD and the event time history from the type representing TEDM. It also adds unanchored primitives and the continuous time domain to its temporal structure. From Figure 23 it can reasonably be concluded that OSAM*/T and T-Chimera are the two least powerful temporal object models since they provide the least number of temporal features. The TOODM model is the most powerful since it provides the most number of temporal features.

The comparison of different temporal object models made in this section shows that there is significant similarity in the temporal features supported by the models. In fact, the temporal features supported by OSAM*/T and T-Chimera are identical. The temporal features of TEDM are identical to those of OSAM*/T and T-Chimera in the temporal structure, temporal representation, and temporal order design dimensions. These commonalities substantiate the need for a temporal framework which combines the diverse features of time under a single infrastructure that allows design reuse.

We also note that temporal object models have not really taken advantage of the richness of their underlying object model in supporting alternate features of a design dimension. They have assumed a set of fixed particular underlying notions of time. From a range of different temporal features, a single temporal feature is supported in most of the design dimensions. As such, not much advantage has been gained over the temporal relational models in supporting applications that have different temporal needs. For example, engineering applications like CAD would benefit from a branching time model, while time series and financial applications require multiple calendars and granularities. The temporal framework proposed in this work aims to exploit object-oriented technology in supporting a wide range of applications with diverse temporal needs.

5 Discussion and Conclusions

In this work the different design dimensions that span the design space of temporal object models are identified. Object-oriented techniques are used to design an infrastructure which supports the diverse notions of time under a single framework. We demonstrate 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.

A similar objective is pursued by Wuu & Dayal [WD92] who 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. The temporal framework presented here subsumes the work of Wuu & Dayal in that it provides the user or database designer with explicit types and properties to model the diverse features of time. Their approach requires significant support from the user, including specification of the temporal schema, which is a complex, and non-trivial task. 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 type system to structure the design space of temporal object models and identify the dependencies within and among the design dimensions helps us simplify the presentation of the otherwise complex domain of time. The framework is extensible in that additional temporal features can be added as long as the relationships between the design dimensions are maintained. The focus in this work is 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 temporal framework also provides a means of comparing temporal objects models according to the design dimensions identified in Section 2.1. This helps identify the strengths and weaknesses of the different models. The diverse features of time are also identified in [Sno95]. 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 work we show how the generic aspects of temporal models can be captured and described using a single framework. In [PLL96] 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++. A toolkit has been developed which allows users/temporal model designers to interact with the framework at a high level and generate specific framework instances for their own applications. The next step is to build query semantics on top of the framework. This will involve addressing issues such as: how the choices of different design dimensions affect the query semantics; what kind of query constructs are needed; what properties should be provided; and how are these properties used, to name a few.

References

[All84] J. F. Allen. Towards a General Theory of Action and Time. *Artificial Intelligence*, 23(123):123–154, July 1984.

[ATGL96] 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.

[BFG97] E. Bertino, E. Ferrari, and G. Guerrini. T_Chimera - A Temporal Object-Oriented Data Model. *Theory and Practice of Object Systems*, 3(2):103–125, 1997.

[BKP86] H. Barringer, R. Kuiper, and A. Pnueli. A Really Abstract Concurrent Model and its Temporal Logic. In *Proc. of the 13th ACM Symposium on Principles of Programming Languages*, pages 173–183, 1986.

[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, 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, June 1993.

[Cho94] J. Chomicki. Temporal Query Languages: A Survey. In D. Gabbay and H. Ohlbach, editors, *Proceedings of the International Conference on Temporal Logic*, pages 506–534. Lecture Notes in Computer Science, Vol. 827, Springer Verlag, July 1994.

[CITB92] 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.

[CJR87] R.H. Campbell, G.M. Johnston, and V.F. Russo. Choices (Class Hierarchical Open Interface for Custom Embedded Systems). *Operating Systems Review*, 21(3):9–17, 1987.

[CK94] S. Chakravarthy and S-K. Kim. Resolution of Time Concepts in Temporal Databases. *Information Sciences*, 80(1-2):91–125, September 1994.

[CMR91] E. Corsetti, A. Montanari, and E. Ratto. Dealing with Different Time Granularities in Formal Specifications of Real-Time Systems. *The Journal of Real-Time Systems*, 3(2):191–215, 1991.

[CPP95] C. Combi, F. Pinciroli, and G. Pozzi. Managing Different Time Granularities of Clinical Information by an Interval-Based Temporal Data Model. *Methods of Information in Medicine*, 34(5):458–474, 1995.

[CPP96] C. Combi, F. Pinciroli, and G. Pozzi. Managing Time Granularity of Narrative Clinical Information: The Temporal Data Model TIME-NESIS. In L. Chittaro, S. Goodwin, H. Hamilton, and A. Montanari, editors, *Third International Workshop on Temporal Representation and Reasoning (TIME'96)*, pages 88–93. IEEE Computer Society Press, 1996.

[CR88] 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, 1988.

[CS93] R. Chandra and A. Segev. Managing Temporal Financial Data in an Extensible Database. In *Proc. 19th Int'l Conf. on Very Large Data Bases*, pages 302–313, August 1993.

[CSS94] R. Chandra, A. Segev, and M. Stonebraker. Implementing Calendars and Temporal Rules in Next-Generation Databases. In *Proc. 10th Int'l. Conf. on Data Engineering*, pages 264–273, February 1994.

[DDS94] 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*, pages 186–195, September 1994.

[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.

[EGS93] O. Etzion, A. Gal, and A. Segev. Temporal Active Databases. In *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, June 1993.

[Flo91] R. Flowerdew. *Geographical Information Systems.* John Wiley and Sons, 1991. Volume 1.

[GLÖS97] I.A. Goralwalla, Yuri Leontiev, M.T. Özsu, and Duane Szafron. Modeling Temporal Primitives: Back to Basics. In *Proc. Sixth Int'l. Conf. on Information and Knowledge Management*, pages 24–31, November 1997.

[GÖS97] I.A. Goralwalla, M.T. Özsu, and D. Szafron. Modeling Medical Trials in Pharmacoeconomics using a Temporal Object Model. *Computers in Biology and Medicine - Special Issue on Time-Oriented Systems in Medicine*, 27(5):369 – 387, 1997.

[HKOS96] 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.

[JF88] R.E. Johnson and B. Foote. Designing Reusable Classes. *Journal of Object-Oriented Programming*, 1(2):22–35, 1988.

[KGBW90] W. Kim, J.F. Garza, N. Ballou, and D. Wolek. Architecture of the ORION Next-Generation Database System. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):109–124, March 1990.

[KKR90] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint Query Languages. In *Proc. of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 299–313, April 1990.

[Kli93] N. Kline. An Update of the Temporal Database Bibliography. *ACM SIGMOD Record*, 22(4):66–80, December 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, June 1992.

[LEW96] 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)*, pages 341–356, October 1996. Proceedings published as Lecture Notes in Computer Science, Volume 1157, Bernhard Thalheim (editor), Springer-Verlag, 1996.

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

[PLL96] 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.

[PM92] N. Pissinou and K. Makki. A Framework for Temporal Object Databases. In *Proc. First Int'l. Conf. on Information and Knowledge Management*, pages 86–97, November 1992.

[Rev90] P.Z. Revesz. A Closed Form for Datalog Queries with Integer Order. In *International Conference on Database Theory*, pages 187–201, 1990.

[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.

[SA85] 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.

[SC91] 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*, pages 431–442, 1991.

[Sci94] E. Sciore. Versioning and Configuration Management in an Object-Oriented Data Model. *The VLDB Journal*, 3:77–106, 1994.

[Sno86] R. Snodgrass. Research Concerning Time in Databases: Project Summaries. *ACM SIGMOD Record*, 15(4), December 1986.

[Sno87] R.T. 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.

[Sno95] 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.

[Soo91] M.D. Soo. Bibliography on Temporal Databases. *ACM SIGMOD Record*, 20(1):14–23, 1991.

[SRH90] M. Stonebraker, L.A. Rowe, and M. Hirohama. The Implementation of POSTGRES. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):125–142, March 1990.

[SS88] R. Stam and R. Snodgrass. A Bibliography on Temporal Databases1. *IEEE Database Engineering*, 7(4):231–239, December 1988.

[TK96] V.J. Tsotras and A. Kumar. Temporal Database Bibliography Update. *ACM SIGMOD Record*, 25(1):41–51, March 1996.

[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, Tempe, USA, February 1992.

[WLH90] K. Wilkinson, P. Lyngbaek, and W. Hasan. The Iris Architecture and Implementation. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):63–75, March 1990.