Report Number: WUCS-91-24

1991-01-01

The Visual Display of Temporal Information

Steve B. Cousins and Michael G. Kahn

The detection of temporal relationships among time-ordered patient data is an important, but difficult, clinical task. Large volumes of computer-stored clinical data offer the possibility of aiding in the early detection of subtle trends and states, but the presence of irrelevant data can obscure relevant findings and relationships. We present a formal system for representing complex temporal data as events on an abstract entity called a time limit. We define five time line operations, SLICE, FILTER, OVERLAY, NEW, and ADD. For each operation, we precisely define the operator's effect on a time line, including exceptions and boundary conditions. IN... **Read complete abstract on page 2.**

The Visual Display of Temporal Information

Steve B. Cousins and Michael G. Kahn

Complete Abstract:

The detection of temporal relationships among time-ordered patient data is an important, but difficult, clinical task. Large volumes of computer-stored clinical data offer the possibility of aiding in the early detection of subtle trends and states, but the presence of irrelevant data can obscure relevant findings and relationships. We present a formal system for representing complex temporal data as events on an abstract entity called a time limit. We define five time line operations, SLICE, FILTER, OVERLAY, NEW, and ADD. For each operation, we precisely define the operator's effect on a time line, including exceptions and boundary conditions. IN addition to our time line, formalism, we describe an interactive environment designed specifically to help humans visualize temporal data. We have developed a database kernel and a graphical user interface that uses our time line formalism and operations to support temporal manipulations. Using our formal system and our visualization environment, we describe two issues in the display and manipulating of temporal data: (1) the temporal granularity problem, and (2) the calendar mapping problem.

The Visual Display of Temporal Information

Steve B. Cousins and Michael G. Kahn

WUCS-91-24

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

Submitted to the journal "AI in Medicine", February 1991.

The Visual Display of Temporal Information

Steve B. Cousins
Michael G. Kahn

Medical Informatics Laboratory
Department of Internal Medicine
Washington University School of Medicine

Correspond with: Steve B. Cousins
Medical Informatics Laboratory
Washington University School of Medicine
660 South Euclid Ave., Box 8121
St. Louis, MO 63110
(314) 362-4322
sbc@informatics.WUSTL.EDU

The Visual Display of Temporal Information

Steve B. Cousins Michael G. Kahn

Medical Informatics Laboratory
Washington University School of Medicine

Abstract: The detection of temporal relationships among time-ordered patient data is an important, but difficult, clinical task. Large volumes of computer-stored clinical data offer the possibility of aiding in the early detection of subtle trends and states, but the presence of irrelevant data can obscure relevant findings and relationships. We present a formal system for representing complex temporal data as events on an abstract entity called a *time line.* We define five time line operations, SLICE, FILTER, OVERLAY, NEW, and ADD. For each operation, we precisely define the operator's effect on a time line, including exceptions and boundary conditions. In addition to our time line formalism, we describe an interactive environment designed specifically to help humans visualize temporal data. We have developed a database kernel and a graphical user interface that uses our time line formalism and operations to support temporal manipulations. Using our formal system and our visualization environment, we describe two issues in the display and manipulation of temporal data: (1) the temporal granularity problem, and (2) the calendar mapping problem.

Keywords: Time line, Visualization

1 Introduction

The chronology of disease symptoms is critical information for patient-specific diagnostic, prognostic, and therapeutic decision making. Temporal issues are so prevalent in the interpretation of clinical data that medical database systems have developed specialized methods to address the unique storage and retrieval requirements of time-varying clinical data. Advances in instrumentation have increased our ability to observe, measure, and record vast quantities of biomedical data. Yet the presence of a large number of data types increases the complexity of detecting the important clinical implications of these measurements.

Most medical databases store data using the time-oriented data (TOD) model [11]. In TOD databases, information is stored as $< attribute, time, value >$ tuples. All laboratory data, physical findings, and therapeutic interventions are represented as events with no meaningful duration, called *point events.* The simplicity and flexibility of the TOD model make it an extremely popular representation method for medical data. Unfortunately, not all medical information fits a point-based representation. Effective diagnosis and therapy planning requires an understanding of temporal trends and clinical contexts in which these patterns occur. For example, it is critical to know if data were measured during a period of illness or while an administered drug was present in therapeutic levels. Since the notion of a drug effect is not a point event, we believe that a medical database must have the ability to represent and manipulate data as both point and interval events. The latter concepts cannot be represented clearly using the traditional TOD model.

We are developing a methodology for formally manipulating and visualizing temporal relationships among biological data. Our objective is to assist users in the interactive exploration of these data and to support computer programs in the automated manipulation of patient data. Our research has approached this goal from both a mathematical and a visualization perspective. We have developed a mathematical formalism, based on the abstract concept called a *time line,* for representing a sequence of events ordered by time. We also have developed a set of mathematical operations that manipulate time lines. In addition, we have developed a visual representation of our mathematical structures and operations. For example, the most intuitive visual representation of a time line is a two-dimensional object, with time on one axis and data and events on the other axis (Figure 1). Because the visualization procedure is separate from the mathematical definition, a time line may be visualized in different ways to solve different problems. We have developed an interactive environment, called a *time line browser,* for displaying and manipulating sets of time lines (Figure 2). Time line operations are applied to time lines in a time line browser to construct new time lines.

The adoption of a more sophisticated temporal data model presents significant new theoretical and practical problems [7,6]. The issue of temporal granularity is of particular interest to our temporal reasoning and visualization research. Temporal granularity is the unit of a time scale appropriate for a given problem-solving context. The problem with temporal granularity is that the set of relevant facts changes whenever a shift in temporal granularity occurs. For example, recording and retrieving information in units of minutes and hours is appropriate in an ICU setting, but weeks or months usually are more appropriate temporal units for the analysis of chronic disease data. Even in an acute setting like the ICU, previously recorded information is manipulated at a different level of temporal granularity than are current data. After discharge from the hospital, the entire ICU stay may be combined into a single interval. We describe one approach to the temporal granularity problem.

A second problem that we discuss is the mapping between "real" or calendar time and "virtual" or relative time. Our formalism allows time lines to be combined in ways which may not map directly to a traditional calendar. For example, our formalism allows a time line to be created by combining two pregnancy events. This may be done to compare temporal features of key events during one pregnancy, such as the appearance of proteinuria or hypertension, to similar events during a current pregnancy. The resulting time line does not occur in "real time"; there is no mapping between the combined events and a calendar. We describe our approach to determining how and when time lines may be mapped to the Julian calendar.

In the remainder of this paper, we develop the mathematical and visual properties of time lines, and discuss applications of the formalism. In Section 2, we formally define the concept of a time line and its basic operations using set theory. In Section 3, we describe how a time line browser allows time lines to be manipulated visually. Sections 4 and 5 describe the use of our formal system and visualization methods to explore the temporal granularity and calendar mapping problems. Finally, Section 6 considers future work.

2 Time Lines

In this section, we describe our representation of events and formally define time lines which contain them. We define a set of basic operations on time lines that result in new time lines. Lastly, we define additional time line operations by composing the basic operators.

2.1 Events

We recognize the need to handle both point and interval events. In our work, interval events are defined in terms of their end-points. Our formalization of time lines addresses only point events; intervals follow because they are defined in terms of points.

Our database schema defines event classes that have similar temporal characteristics. This grouping allows common properties and behaviors to be associated with each class of temporal concepts from the domain that is modelled in the database. Although these distinctions among events have no bearing on our mathematical treatment of time lines, they are at the heart of the visualization aspects of this work, and are integral to our solution to the temporal granularity problem. In Table 1, we describe the basic event classes and their default visual representations. A taxonomy of event classes is used to describe temporal and atemporal properties and behaviors associated with events contained in a patient's database. Table 1 also presents the implementation features of the three basic temporal classes defined in our current system. *Simple events* are analogous to the $<attribute, time, value>$ tuples in TOD databases. They represent point events. *Complex events* represent point events whose time of occurrence has been abstracted to a single point. *Intervals* represent events with temporal duration.

Our taxonomy distinguishes between point and interval event classes because the properties and behaviors associated with each class differ significantly. Point events are atomic; they contain no additional events. Points occur before, during, or after other points. Intervals may contain both points and subintervals. Intervals also may be overlapped by, concurrent with, contained in, or contiguous to other intervals [1]. We may need to retrieve all events which occur within the duration of a specified interval event, but it makes no sense to make the same query of a point event. We also note that the display behavior of point and interval events are different. Interval events need to display temporal duration, whereas point events do not (Figure 1, Section 3).

The complex event class is distinguished from the point event and interval event classes as a modelling convenience. Certain events, such as intramuscular injections, are simply point events from a clinical point of view; the duration of the injection may reasonably be considered to be zero. Other events have meaningful duration, but that duration may not be recorded because it typically is not considered during clinical reasoning. For example, hypoglycemic symptoms have a meaningful duration, usually lasting a few minutes, but we reason about them only by their existence, normally ignoring their duration. For clinical management, diabetic patients are asked to record *when* they experienced hypoglycemic symptoms, but not how long they lasted.

2.2 Formal Definition of Time Lines

Time lines contain a set of events. Formally, a time line is a tuple $< E, M >$, where E is a finite set of events containing at least the special *null event* e_Φ, and M is a measure function $M : E \rightarrow \mathcal{R}^+$. The measure function M assigns a temporal offset to each event in E. By definition, $M(e_\Phi) = 0, \forall e_i \in E, M(e_\Phi) \leq M(e_i)$, meaning that no event may come before the null event. One special time line is the *null time line*, $TL_\Phi =< \{e_\Phi\}, M >$, which consists of only the null event.

Intuitively, a time line is a line segment with e_Φ as its leftmost boundary, some e_n such that $M(e_n)$ is maximal as its rightmost boundary, and all other events placed in between according to the temporal ordering imposed by M. The measure function imposes a total ordering on E. Our formalism assumes a standard unit of time for the measure function. The choice of a particular unit (e.g. seconds, minutes, days) is arbitrary. In practice, the choice of a unit measure for manipulating multiple time lines should be standardized. For our implementation, the unit measure we have selected is seconds.

In the rest of this section, we formally define a set of time line operations. Informally, SLICE corresponds to removing events from one or both ends of a time line, thereby reducing its size. FILTER removes events that do not satisfy an arbitrary predicate from the time line. OVERLAY corresponds to combining two time lines into one. One event in each time line is specified as the aligning event; the new time line contains all of the events of the old ones aligned on the specified events. NEW creates a new, empty time line, and ADD allows an event to be added to an existing time line. We show how other operations may be defined in terms of our primitive operations.

2.3 Slice

SLICE removes events from one or both ends of a time line (Figure 3A). A time line $TL =< E, M >$, sliced from e_1 to e_2 ($e_1, e_2 \in E$ and $M(e_1) \leq M(e_2)$), yields a new time line

$$TL' =< E', M' >= \text{SLICE}(TL, e_1, e_2) \tag{1}$$

where

$$E' = \{e \in E | M(e_1) \leq M(e) \leq M(e_2)\} \cup \{e_\Phi\} \tag{2}$$

$$M'(e \in E') = \begin{cases} 0 & \text{when } e = e_\Phi \\ M(e) - M(e_1) & \text{otherwise} \end{cases} \tag{3}$$

Equation 1 describes the structure of a call to SLICE, using standard functional notation. Equation 2 adds all events whose measure function puts them between e_1 and e_2 inclusive, to the new time line. The way this definition is structured implies that the events between e_1 and e_2 will exist in both time lines—E' is *not* made up of copies of the events from E. Equation 1 also guarantees that the null event e_Φ is in E'. Equation 3 defines a new measure function for the new time line, which maintains the same relative temporal offsets among events in TL' as in TL. Note that the definition of SLICE permits $e_1 = e_2$ (resulting in a time line with only the events e_Φ and e_1), and that $(\forall TL)\text{SLICE}(TL, e_\Phi, e_\Phi) = TL_\Phi$, i.e. slicing the null time line gives a null time line.

2.4 Filter

FILTER (Figure 3B) removes all events not satisfying an arbitrary predicate P from a time line $TL =< E, M >$. By definition, the null event e_Φ cannot be removed by any predicate. The new time line is

$$TL' =< E', M >= \text{FILTER}(TL, P) \tag{4}$$

where

$$E' = \{e \in E | P(e)\} \cup \{e_\Phi\} \tag{5}$$

Equation 5 applies the predicate P to all events except e_Φ. The null time line TL_Φ is generated whenever a predicate P removes all events (other than e_Φ) from the original time line TL.

2.5 Overlay

OVERLAY puts all of the events in two time lines into a single one, so that an event from the first time line and one from the second time line have the same measure function in the new time line

(Figure 3C). A subtle complication occurs in the OVERLAY operation that does not occur in any other temporal operation. If both original time lines have an event in common, it is undesirable to have that event duplicated in the new time line at the same point in time, since then even simple operations such as counting the events in the new time line would not act as the user expects. On the other hand, if the alignment operation does not happen to align the common event, the measure function for that event could have two potential values. Because it makes no sense to give the same event two different measure function values, duplication of that event is required (Figure 4). We use an operation called **copy** to do the necessary duplication in Equation 10. We define **copy** for events as $\mathbf{copy}(e) = e'$ such that $(\forall P)P(e) \leftrightarrow P(e')$ and $e \neq e'$ where P is an arbitrary predicate defined over events. After **copy**, the events e and e' have the same properties but are not the same event.

OVERLAY combines two time lines

$$TL_1 = < E_1, M_1 > \tag{6}$$

$$TL_2 = < E_2, M_2 > \tag{7}$$

into a new time line,

$$TL' = < E', M' > = \text{OVERLAY}(TL_1, e_1, TL_2, e_2) \tag{8}$$

aligning the time lines on $e_1 \in E_1$ and $e_2 \in E_2$, where

$$M_1(e_1) \geq M_2(e_2) \tag{9}$$

$$\begin{aligned} E' \;=\; & E_1 \cup E_2 \cup \\ & \{\mathbf{copy}(e) | (e \in E_1 \cap E_2 \setminus \{e_\Phi\}) \wedge \\ & \quad (M_1(e_1) - M_1(e) \neq M_2(e_2) - M_2(e))\} \end{aligned} \tag{10}$$

Equation 10 adds to E' all events in E_1 and E_2 and copies of events which occur in both E_1 and E_2 but which will not be combined in E'. To simplify the definition of M', we define the following:

$$\mathbf{shift} \equiv M_1(e_1) - M_2(e_2) \tag{11}$$

$$M_2(copy(e)) \equiv M_2(e) \tag{12}$$

Equation 11 gives a name to the temporal offset between the two time lines. Note that the precondition in Equation 9 ensures that **shift** will not be negative. An implementation could relax the restriction in Equation 9 by testing for it and swapping the time lines if necessary to make it hold. Equation 12 defines the measure function for copied events. With these definitions, we define the new measure function for TL' as:

$$M'(e \in E') = \begin{cases} M_1(e) & \text{when } e \in E_1 \\ M_2(e) + \textbf{shift} & \text{when } e \in E_2 \wedge e \notin E_1 \\ M_2(e) + \textbf{shift} & \text{when } e \notin E_1 \cup E_2 \end{cases} \tag{13}$$

In Equation 13, the first case assigns a value to the new measure function for events which come from TL_1, including those events which occur in both TL_1 and TL_2. The second case handles events from TL_2, except those which have already been given a value in M' because they are also in TL_1. The third cases handles the remaining events in E', which are only those events resulting from a copy.

2.6 New

The NEW operation constructs a null time line (Figure 3D):

$$TL_\Phi = < \{e_\Phi\}, M > = \text{NEW}() \tag{14}$$

2.7 Add

ADD is used to add an event e to a time line $TL = < E, M >$ at offset t:

$$TL' = < E', M' > = \text{ADD}(TL, e, t) \tag{15}$$

where

$$e \notin E \tag{16}$$

$$E' = E \cup \{e\} \tag{17}$$

$$M' = M \cup \{< e, t >\} \tag{18}$$

Formally, we treat the function M as a set of ordered pairs, and add elements to the domain over which M is defined. The restriction of Equation 16 ensures that M remains a proper function after the operation.

2.8 Composite Operations

The operations we have defined are intended to provide minimal functionality for time lines. Other operations can be defined in terms of these primitive operators. For example, concatenation can be performed as a special case of OVERLAY by using the last event of the first time line and the first event of the second time line as the aligning events. A time line can be copied by slicing from its first event to its last event. Even the time of an event on a time line can be changed:

$$\text{MOVE}(e, TL) \equiv \text{ADD}(\text{FILTER}(TL, (\lambda x)(x \neq e)), e, \text{newtime}) \tag{19}$$

Here $\text{FILTER}(TL, (\lambda x)(x \neq e))$ is simply a function to remove e from TL.

3 Time Line Browsers

Although time line operations have the conceptual power to manipulate time lines as mathematical abstractions, they do not specify the visual behavior of time lines. We have developed an editor for time lines, called a *time line browser*, which implements time line operations as well as additional operators specific to time line browsers. In this section, we describe the time line browser and how it provides visual access to the various time line operations.

In order to interact with time lines graphically, we need a 2-dimensional visualization of them (Figure 1). The only required dimension is time, so we are free to use the second dimension at our discretion. Some types of events have integer- or real-valued components. In these cases, the second dimension often is used to plot the value of these components. Other event types are informational, and the second axis is used only to avoid overwriting co-temporal events. Small circles in Figure 1 represent blood glucose readings, positioned vertically by value. The horizontal positions of the X-ray icon and the speakers indicate when an X-ray was taken or when voice-commentaries were made to the record, respectively. The widths of the boxes denoting interval events indicate their temporal duration.

3.1 General Structure of Time Line Browsers

Time lines are placed in rows in a time line browser (Figure 2). The order of the time lines is determined by the user. Two adjacent time lines may be aligned (visually synchronized) to indicate temporal synchronization. When not aligned, adjacent time lines are separated by a dashed line.

At any given time, a subset of the objects in a time line browser (time lines and the events they contain) are *selected* [2]. In general, objects are selected by clicking on them. An entire time line is selected by clicking on its border. Multiple events are selected by holding the *add to selection* modifier key down (typically the shift key). An entire class of events is selected by clicking on any instance of the event class with the *class* modifier key down.

3.2 Visualizing Time Line Operations

Just as time lines have both a mathematical definition and a visual representation, we have developed visual analogs of the time line operations.

Visually, SLICE is performed by selecting a range of events and choosing the SLICE option from a menu. The first event in the selection is taken as e_1 and the last event selected is taken as e_2. The new time line is inserted in the time line browser immediately after the one from which it was sliced, and is aligned with the original time line.

Visualizing FILTER is complicated by the need to specify the predicate P in Equation 4. Because all events are members of some event-class, this problem is solved best in general at the event-class level. We provide shortcuts for a few simple predicates. In particular, the predicate $P_C(e) \equiv (e \notin C)$ (for an arbitrary event class C) which removes all events in class C, is performed by selecting a class of events, and then striking the delete key. Similarly, a specific event is removed by selecting it and striking the delete key, which invokes the operation $\mathrm{del}(e') \equiv \textsc{filter}(TL, P)$ where $P_{\mathrm{del}}(e) \equiv e \neq e'$. In general, however, selecting an event and choosing FILTER from a menu invokes a class-specific method for constructing predicates relevant to the event's class. Although, the formal definition of FILTER actually returns a new time line, applying FILTER in a time line browser visually replaces the current time line with the new time line created by the operation in order to provide the feeling that the user is directly manipulating time lines in the time line browser.

OVERLAY is performed by dragging the aligning event of one time line onto the aligning event of a second time line. The resulting time line visually replaces the second time line in the time line browser.

Finally, a new (null) time line is created by choosing the NEW command from a menu. Alternately, a new time line can be created from existing time lines by taking an arbitrary SLICE from a time line, and then using FILTER to remove all events.

3.3 Visualizing Time Line Browser Operations

The utility of the time line browser is greatly enhanced by adding browser-level operations to the collection of time line operations defined above. Unlike time line operations, time line browser operations are applied to one or more time lines contained in a time line browser. We describe operations which allow the user to ALIGN time lines in the time line browser, to change the time SCALE (temporal granularity), and to MARK events.

3.3.1 Align

ALIGN is used to arrange two adjacent time lines so that appropriate events are lined up vertically (Figure 2). Typically, ALIGN is used to give two time lines a common temporal basis. For example, two time lines representing the same period of calendar time would be aligned on a common day, or two pregnancies may be aligned on the dates of conception. Time lines can be moved within time line browsers, but it may be necessary to break their alignment with other time lines in order to do so.

3.3.2 Scale

SCALE refers to the amount of time represented by a unit of space on the horizontal axis (Figure 6). To avoid visual confusion, all time lines within a single time line browser are drawn to the same temporal scale. For example, the scale in Figure 1 is approximately 1 day = 1 inch. Zooming out to a coarser temporal granularity and zooming in to a finer temporal granularity are implemented as changes in the SCALE of a time line browser. We discuss the implications of changes in temporal scale in the context of the temporal granularity problem in Section 4.

3.3.3 Mark

Selecting a set of events and choosing MARK temporarily alters the visual characteristics of the selected events. For example, the marked events may have their shapes modified, or on a color monitor the events may all be drawn in a new color. In Figure 2, vacation events are marked with an "×" rather than a "o". Although a MARK is an attribute of a time line browser, its effect is to change the visual characteristics of a time line.

3.3.4 Other operations

The number of functions desired in a time line browser may eventually approach the number of operations in modern text editors, and we do not intend for those listed here to constitute a complete set. ALIGN, SCALE, and MARK are some of the more unusual operators.

For practical use of a time line browser, time lines may be selected and saved to a file individually, or complete time line browsers may be saved. If the entire time line browser is saved, all positioning information is retained in the file. Similarly, time lines can be loaded from files into new time line browsers, and time line browsers which have been saved can be restored.

3.4 A Time Line Browser for Diabetes

Using the analysis and display of diabetes patient data as our application area, we have implemented a time line browser prototype which displays a patient's medical history as a time line. We have defined application-specific subclasses of the three basic temporal classes to encode temporal entities typically found in diabetes data (Figure 5). Application-specific classes which are interval-based, such as illness-intervals, are subclasses of interval, while classes which are point-based are subclasses of either simple-event or complex-event. This prototype system supports the time line manipulations we have described. One key feature for visualizing large data sets collected over a long period of time is the user's ability to zoom in and out to different levels of temporal abstraction (Figure 6).

In Figure 2, we show a time line browser for a hypothetical diabetic patient. The uppermost time line in the time line browser represents the output of the patient's diabetic logbook[1]. Note the run of markedly hyperglycemic readings (indicated by "×"s) on Monday through Wednesday. The second time line is a *slice* from the patient's personal calendar (a form of time line), which is temporally *aligned* with the logbook. When available, a calendar can give a physician additional information about the patient's lifestyle that the medically-oriented logbook does not include. In this case, the calendar indicates that the patient visited family members during those three days, which in turn suggests the hypothesis that the patient had either more food or less exercise than usual due to the break in his normal schedule. The third time line in Figure 2 is separated from the first two by a dashed line, indicating that it is not temporally aligned with them. This time line is a *modal day*—created by *overlaying* a whole week's worth of days. The modal day is popular in the domain of diabetes management because it gives an indication of the range of values at different times during the day.

4 Temporal Granularity

Temporal granularity is the unit of a time scale appropriate for a given problem-solving context [3]. Exact seconds do not matter if the concept of interest ranges over years, but seconds become important when that concept evolves over minutes. The *temporal granularity problem* is that as the temporal granularity grows, the number of entities to be considered grows. The visual aspect of this problem is that as the temporal granularity grows, the visual clutter on the screen grows. To keep a decade's worth of data from becoming a black band on a time line, we propose *temporal granularity heuristics* which attempt to determine which classes of events to display dynamically as the temporal granularity changes.

Temporal abstraction is an effective mechanism for combining smaller temporal entities into larger, but less detailed, concepts. Abstraction simplifies retrieving and reasoning by combining multiple features into a single entity. Temporal decomposition is the inverse operation of temporal abstraction. In temporal decomposition, the entities contained within a larger temporal abstraction are available for more refined reasoning. The number of entities to manipulate is increased by decomposing an abstraction. We believe that temporal abstraction and decomposition are the expression of a powerful heuristic used by humans to focus only on features that are relevant to solve a specific problem [5].

[1] Most diabetic logbooks are recorded by hand today, and do not usually have this variety of information. However, hand-held electronic logbooks are being developed, which will make this information much more readily available to analysis programs in the near future.

We seek to develop an automated method that suppresses specific information in some problem-solving contexts, but makes these details available when appropriate. Because of the vague nature of "when appropriate", the best we can hope for is a heuristic. The purpose of a temporal granularity heuristic is to minimize the extraneous detail that always exists at a particular temporal granularity. By eliminating the visual clutter caused by unwanted detail, the user of the browser can perform time line manipulations with only those concepts that are required to solve his current analysis problem. Event classes in our system can use temporal abstraction to suppress details when the temporal granularity changes and makes them irrelevant.

Our temporal granularity heuristics encode the information about relevant temporal granularities in each class of events. This architecture provides a simple way to customize various temporal granularity heuristics. In this way, objects can change behaviors depending on the current temporal granularity. We use this property to modify an object's response to queries and for the visual display of information. For example, blood glucose readings are individually displayed when the temporal granularity is such that only a few weeks are displayed, but are suppressed when the granularity increases so that many months or years are displayed.

One way to implement temporal abstraction and decomposition is to define a subclass of interval called a *sub-time-line.* Using this subclass, a hierarchy of time lines can be represented, with each sub-time-line containing events which have been abstracted from the parent time line. Given such a hierarchy, temporal decomposition is performed by overlaying the sub-time-line onto the original time line, creating a time line which contains the events of both time lines. Temporal abstraction is implemented by creating sub-time-lines containing the abstracted events and replacing the abstracted events with the sub-time-line in the original time line. Temporal abstraction can be achieved with our current set of time line operators.

5 Mapping Time Lines onto a Calendar

A key feature of our time line definition is the clear distinction between a temporal ordering and calendar time. When two separate days are sliced from a single time line and are overlayed to form a new time line representing a composite day, the new time line no longer has a direct relation to the Julian calendar (What would its date be?). In order to separate these ideas, we need to suspend the notion of calendar time, but keep the proper temporal ordering. Because of the definition of the measure function M, our basic time line formalism contains only the notion of temporal ordering of events. We refer to time lines which can be directly mapped to the Julian calendar as *grounded time lines.*

A time line is grounded when it has a direct mapping to the Julian Calendar. Let $C =< E_C, M_C >$

be the Julian Calendar starting at some arbitrary (but finite) point in time. All events which occur in the world are included in C. Time line $TL = < E, M >$ is grounded if

$$\exists s \in R^+ \text{ such that } \forall e \in E(e \in E_C \wedge M(e) + s = M_C(e)) \tag{20}$$

Note that s (for shift) is the offset of the grounded time line from the beginning of the Julian Calendar.

The modal day plot in Figure 2 is an example of an ungrounded time line. It contains a temporal ordering, but does not map onto any particular part of the calendar. The modal day plot is formed by filtering all events except blood glucose readings from the first time line, marking the values on the three-day weekend, slicing the resulting time line into single day time lines, and then overlaying the single day time lines onto each other. This type of plot is very popular among physicians managing diabetics because it aggregates a large amount of data in a meaningful way.

It follows that the null time line is always grounded, and that the Julian Calendar itself is grounded. Any time line created with SLICE or FILTER from the Julian Calendar is also grounded. Ungrounded time lines are always a result of either creating a new time line with NEW, and then using ADD, or performing an OVERLAY operation.

6 Discussion

The operations described in this paper provide a basic framework for manipulating temporal data. We conclude with a discussion of the implementation, scalability, and future extensions of this work.

Our prototype time line browser is designed for diabetes data management. Data common to this problem domain are blood glucose measurements, meals, and insulin injections (point events), and exercise and unusual events such as hypoglycemia or illness (interval events). Although our initial prototype addresses issues in diabetes data storage and retrieval, our system is applicable to a much wider range of applications, both inside and outside of the domain of medicine.

We use object-oriented programming methods [4,8] to implement the temporal classes and to store data instances. A class object stores properties that are common to all instances of that class. Programming procedures associated with each class object implement behaviors that are common to all class instances. Operations among database entities are performed by sending messages to data objects in the database. Figure 5 shows the object hierarchy for events in our diabetes time line browser. The root of the hierarchy *events,* contains a default action for most defined behaviors. In the object-oriented paradigm, a function is defined to perform a default behavior. Subclasses specialize the default behaviors by defining functions with the same name as the parent class. These specialized behaviors then become the default behavior for that subclass and any child subclass

(Table 2).

Accommodating large quantities of data is a serious issue in any real visualization system. We minimize this problem through the use of data abstraction techniques which aggregate irrelevant data. For example, when reasoning about a previous hospitalization, it is often reasonable (and desirable) to suppress minor details of the hospital course. Abstraction aggregates detailed information into a less detailed composite concept. This technique represents a trade-off between the necessary retention of all medical data and the time-consuming retrieval of data in a medical problem-solving environment. We propose to remove the primary data from the active portion of the medical record, leaving only a hospital interval abstraction, which has a pointer to a long-term storage device (such as a tape label) that contains the detailed data. If the hospital abstraction interval ever is queried for more details, the object's response might be to request that the appropriate magnetic tape be mounted so that the requested details could be re-incorporated into the patient's active history.

We have not dealt with incorporating temporal uncertainty into our time line operations. Others have shown the general solution to temporal uncertainty to be an NP-hard problem [1,10]. Our framework allows one to specify bounds on the occurrence of events, but we have not dealt with how to resolve that temporal uncertainty. For example, assume we have two events regarded as having occurred at 11:55 AM. When viewed at a very fine temporal granularity, it becomes clear that the events could have occurred at different times. In these cases, we add error bars to the event's visual representation to indicate that, *at this temporal granularity*, the exact location of the event on the screen is *not* indicative of its exact temporal location.

There are other visual representations of time in common use which we believe our system will be able to handle, but which we have not yet implemented. For example, we have examined a large number of temporally-oriented statistical graphics from Tufte's book [9], and are confident that our framework can handle them. There are two commonly-used visual representations that we do not currently handle. One is the "traditional" time line: A single line with other lines perpendicular to it, such as in Figures 3 and 4. Another useful visual representation displays time lines vertically (as in a patient chart or date book). One way to handle these alternative representations, is to require event classes to respond to new display messages, such as "traditional draw". Unfortunately, this requirement would add to the burden of programmers of event classes, making them define a separate draw routine for each possible visual representation. We have a tentative design which we believe will allow us to draw these alternative time line formats without placing such a burden on event class programmers. At the current time, our prototype draws only time lines in the format shown in Figures 1 and 2.

It will take more than fast display algorithms to satisfy the needs of physicians to visualize patient data. We believe that having a powerful, concise, and intuitive set of operations is absolutely necessary to allow clinicians to perform more thorough analysis of temporal data. Therefore, we have developed formal specifications that precisely define the temporal characteristics of all time line operations. Each definition describes both the alterations in a time line imposed by the operation and a description of a visual format for displaying the results of the operation. We seek to develop a complete calculus of time line manipulations and visualizations that can be generalized to other sources of time-oriented biological data. With precise semantics and a complete toolbox of operators, we believe the time line concept will become a useful visualization approach to browsing biomedical data.

Acknowledgements

This work was partially supported by the Washington University Center for Intelligent Computer Systems, by a series of generous hardware grants from Apple Computer, Inc., and by the Diabetes Research Training Center at Washington University (NIH grant 5 P60 DK20579). We appreciate the comments of Charlene Abrams and Mark Frisse on previous versions of this manuscript.

References

[1] James F. Allen. Maintaining knowledge about temporal intervals. *Communcations of the ACM*, 26:832–843, 1983.

[2] Apple Computer, Inc. *Inside Macintosh.* Addison Wesley, Reading, MA, 1985.

[3] Steve B. Cousins, Michael G. Kahn, and Mark E. Frisse. The display and manipulation of temporal information. In *Proceedings, Symposium on Computer Applications in Medical Care*, New York, NY, November 1989. IEEE Computer Society Press. Also available as Washington University Computer Science technical report wucs-89-48.

[4] Brad J. Cox. *Object-Oriented Programming: An Evolutionary Approach.* Addison-Wesley, Reading, MA, 1986.

[5] Benjamin Kuipers. Abstraction by time-scale in qualitative simulation. In *Proceedings AAAI-87*, San Mateo, CA, 1987. Morgan Kaufmann.

[6] Richard Snodgrass. Temporal databases: Status and research directions. *SIGMOD Record*, 19:83–89, Dec. 1990.

[7] Richard Snodgrass and Ilsoo Ahn. Temporal databases. *IEEE Computer*, 19:35–42, 1986.

[8] Mark Stefik and Daniel G. Bobrow. Object-oriented programming: Themes and variations. *AI Magazine*, pages 40–62, 1986.

[9] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT, 1983.

[10] Marc Vilain and Henry Kautz. Constraint propagation algorithms for temporal reasoning. In *Proceedings IJCAI-86*, pages 377–382, San Mateo, CA, 1986. Morgan Kaufmann.

[11] Gio Wiederhold, James F. Fries, and Stephen Weyl. Structured organization of clinical data bases. In Donal A. Meier and Stephen W. Miller, editors, *AFIPS Conference Proceedings*, volume 44, pages 479–485, Montvale, NJ, 1975. AFIPS Press.

Table 1: Event types and their visual representations

Time Line Event	Description	Default Visual Depiction	Temporal Fields
Simple Event	Non-decomposable event	$\circ$	Date
	Simple event occurring at any time in the interval denoted by the line segment	—$\circ$—	(Future Work)
Complex Event	Decomposable, but typically aggregate event	$\boxed{\circ}$	Date, Interval
	Decomposable event occurring at some unknown time in the interval denoted by the line segment	—$\boxed{\circ}$—	(Future Work)
Interval	Event with a significant duration	▭	Event, Event
	Interval occurring at any time in the larger interval denoted by the line segment	—▭—	(Future Work)

Table 2: Sample event-object default and specialized behaviors.

Object Class	Representative Action		
	Draw	Y-location	Double-click
Blood Glucose	Small circle	BG value	Show dialog box
X-Ray	X-Ray icon	0.75	Show XR picture
Event (default)	Small circle	0.50	No action

Figure 1: A time line displaying various event classes that might be found in a diabetic patient record. The small circles represent blood glucose readings, positioned vertically by value and horizontally by time of measurement. The horizontal positions of the X-ray icon and the speakers indicate when an X-ray was taken or when voice-commentaries were made to the record, respectively. Boxes denote interval events; the width of a box indicates an interval event's temporal duration.

Figure 2: A time line browser is an editor for time lines. This time line browser displays data from a diabetic patient in three time lines: (a) a diabetes logbook record; (b) a portion of the patient's personal calendar; (c) a modal day plot. "B L D N" is a short-hand for the time of day: Breakfast, Lunch, Dinner, or Nighttime.

Figure 3: Operations on time lines. We use the more traditional visual representation for time lines in this figure for clarity.

Figure 4: Two time lines showing the effects of an OVERLAY operation when the time lines have events in common (events b and c). (a) When the OVERLAY operation places the same event from two time lines in exactly the same place, only one event results. (b) If the repeated events must occur at two times in the resulting time line, the repeated events are copied. The events b' and c' are generated by **copy**(e).

Figure 5: A taxonomy of temporal classes in diabetes data.

Figure 6: Two time line browsers displaying different levels of temporal granularity. Events which are relevant at one temporal granularity are removed at other temporal granularities, where they are less relevant and would only clutter up the display.

Table 2

Patient: Joe Patient

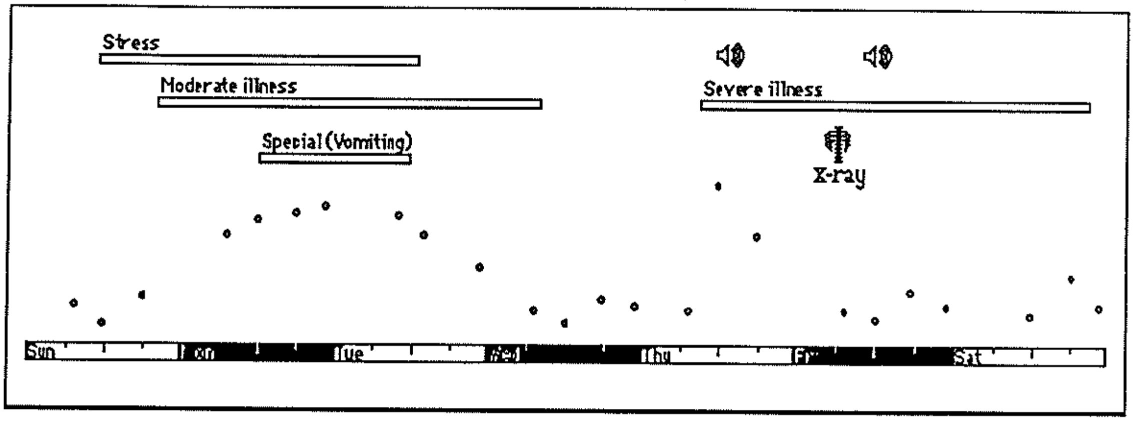
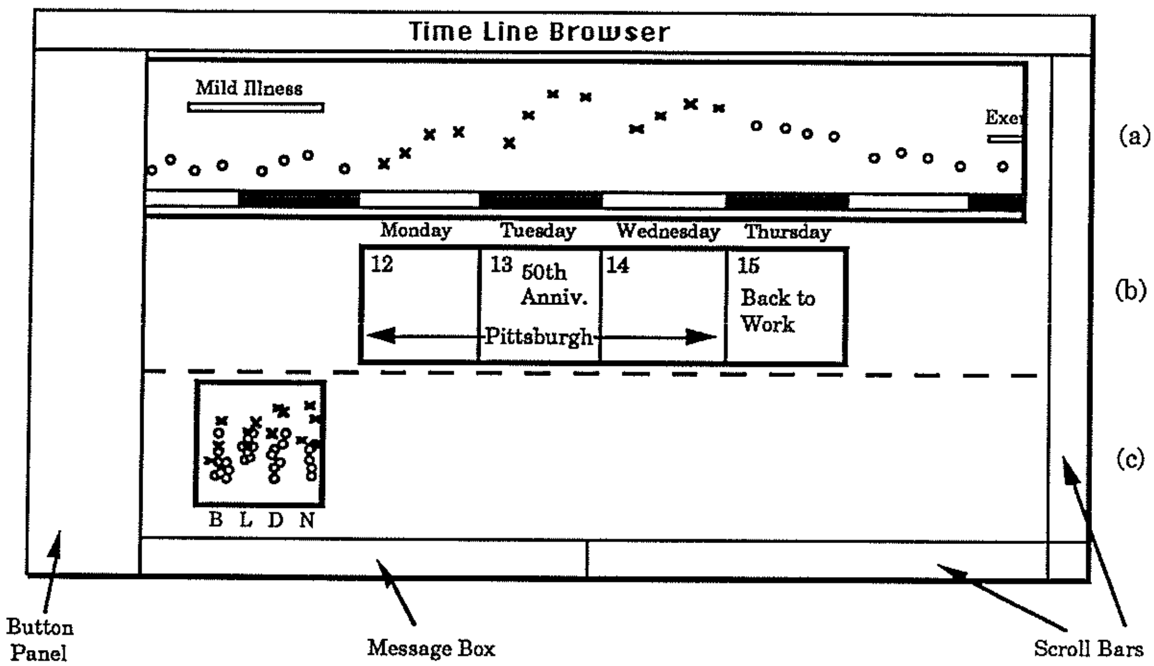


Figure 1



Time Line Browser
Mild Illness
Exe
Monday
Tuesday
Wednesday
Thursday
12
13 50th Anniv.
14
15 Back to Work
Pittsburgh
B L D N
(a)
(b)
(c)
Button Panel
Message Box
Scroll Bars

Figure 2

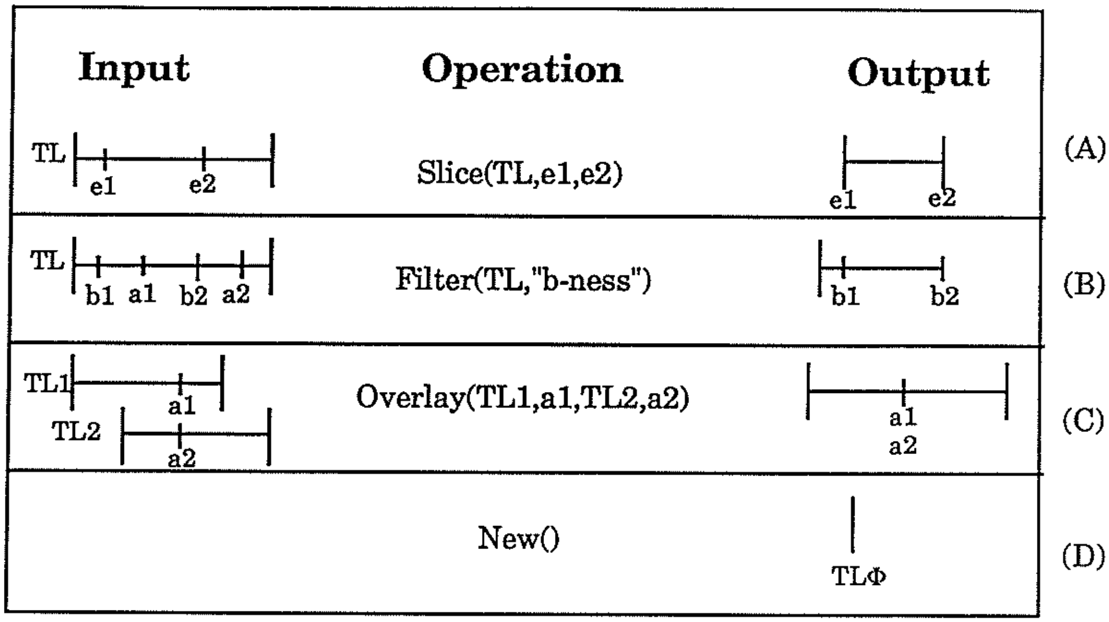

Input
Operation
Output
TL
e1
e2
Slice(TL,e1,e2)
e1
e2
(A)
TL
b1 a1 b2 a2
Filter(TL,"b-ness")
b1
b2
(B)
TL1
a1
TL2
a2
Overlay(TL1,a1,TL2,a2)
a1
a2
(C)
New()
TLΦ
(D)

Figure 3

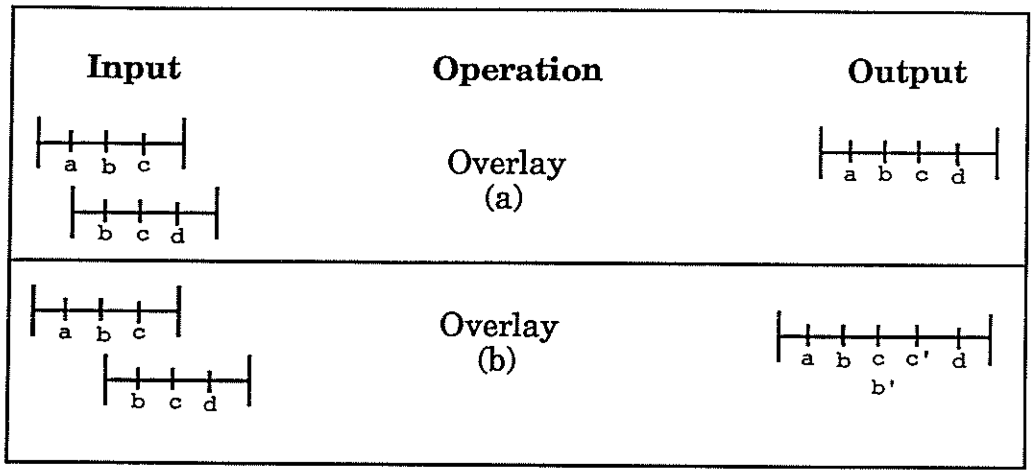

Input
Operation
Output
a b c
b c d
Overlay
(a)
a b c d
a b c
b c d
Overlay
(b)
a b c c' d
b'

Figure 4

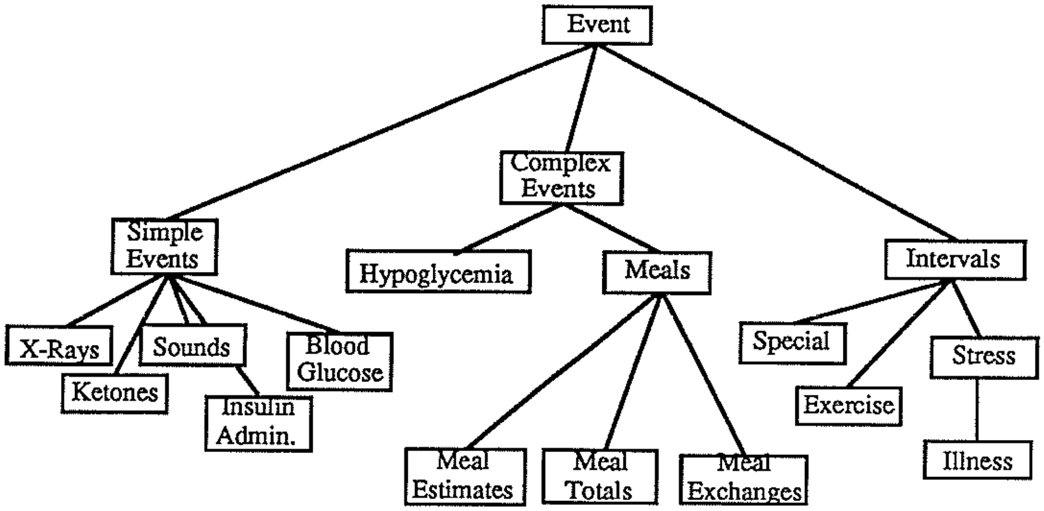

Event
Simple Events
Complex Events
Intervals
X-Rays
Ketones
Sounds
Insulin Admin.
Blood Glucose
Hypoglycemia
Meals
Meal Estimates
Meal Totals
Meal Exchanges
Special
Exercise
Stress
Illness

Figure 5

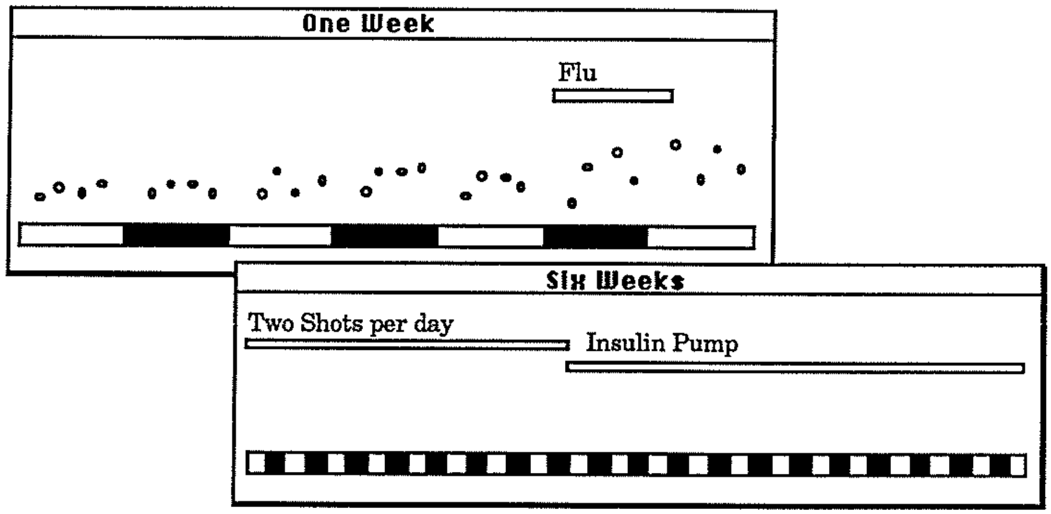


Figure 6