TimeBench: A Data Model and Software Library for Visual Analytics of Time-Oriented Data

Alexander Rind, Tim Lammarsch, Wolfgang Aigner, Bilal Alsallakh, and Silvia Miksch

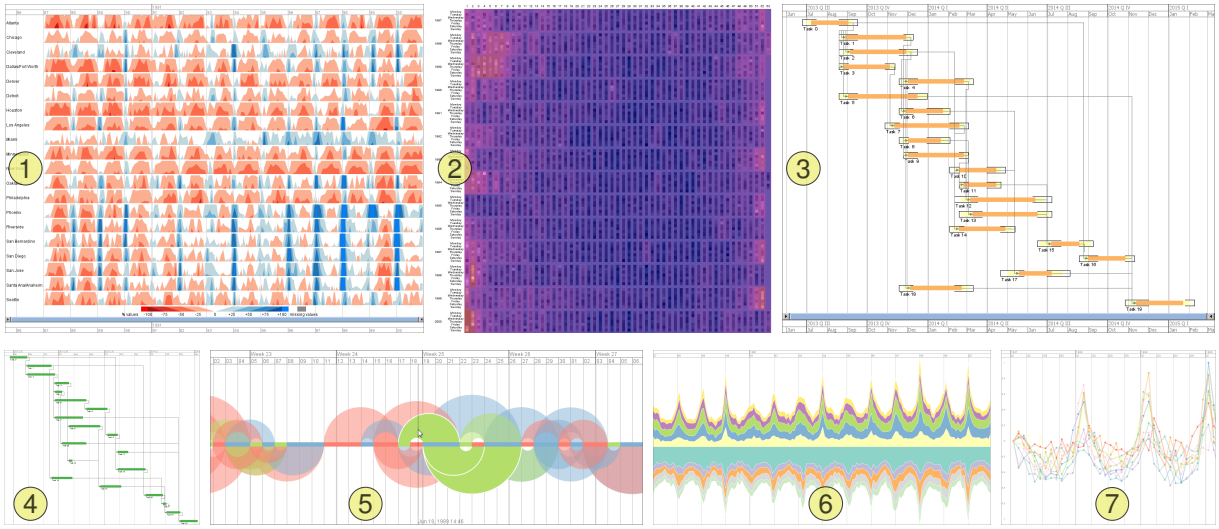

Fig. 1. Application examples built using TimeBench: (1) Monthly health data of 20 cities over 14 years in a Horizon Graph [44]; (2) 14 years of daily health data in a GROOVE visualization [39]; (3) a project plan using the PlanningLines metaphor [3]; (4) a project plan as a Gantt chart [28]; (5) an Arc Diagram [50] showing relationships between events of three categories; (6) a ThemeRiver visualization [29]; (7) multiple line plots with indexing [9]. Health data from the NMMAPS study [42] is used in (1), (2), (5), (6), and (7).

Abstract—Time-oriented data play an essential role in many Visual Analytics scenarios such as extracting medical insights from collections of electronic health records or identifying emerging problems and vulnerabilities in network traffic. However, many software libraries for Visual Analytics treat time as a flat numerical data type and insufficiently tackle the complexity of the time domain such as calendar granularities and intervals. Therefore, developers of advanced Visual Analytics designs need to implement temporal foundations in their application code over and over again. We present *TimeBench*, a software library that provides foundational data structures and algorithms for time-oriented data in Visual Analytics. Its expressiveness and developer accessibility have been evaluated through application examples demonstrating a variety of challenges with time-oriented data and long-term developer studies conducted in the scope of research and student projects.

Index Terms—Visual Analytics, information visualization, toolkits, software infrastructure, time, temporal data

1 Introduction

Time-oriented data is ubiquitous in many domains as for example medicine, business, engineering, and security. Time itself is an inherent data dimension that is central to the tasks of revealing trends and identifying patterns and relationships in the data. Time and time-oriented data have distinct characteristics that make it worthwhile to treat such data as a separate data type [2, 5]. This view has also been expressed earlier by Shneiderman [45] as well as by Card and Mackinlay [15]. As example for such characteristics, time-oriented data can be given for either a time point or a time interval. While intervals can easily be modeled by two time points, they add the complexity of 13 qualitative temporal relations [4]. Also, intervals of validity may be relevant to domain experts but might not be explicitly specified in the data. Another characteristic of time is its calendar aspect: we often interpret and reason about time by using a calendar whose time units are essential for this reasoning. However, calendars have complex structures: in the Gregorian calendar the duration of a month varies between 28 and 31 days and a single week can overlap with two different months and even two different years. Furthermore, available data might be measured at different levels of temporal precision. Some patterns in time-oriented data emerge when a cyclic structure of time is assumed, as with traffic volume by time of day or unemployment rate by season. In other cases, analysts need to filter out such effects in order to understand long-term trends. Additionally, analysts might be interested in comparing developments in the data that do not cover the same period of time. For such comparisons, the analysis is usually focused on relative time, according to some sentinel events.

Therefore, visualization and analysis techniques need to address the characteristics of time. Time is comprehended differently than other dimensions by analysts and might influence other variables and physical dimensions. Therefore, special methods for time-oriented data are necessary in order to reveal trends and identify patterns that might be hidden if time is treated merely as a quantitative dimension. Many methods and applications exist for the visualization of time-oriented

• *Alexander Rind, Tim Lammarsch, Wolfgang Aigner, Bilal Alsallakh, and Silvia Miksch are with Vienna University of Technology, Institute of Software Technology & Interactive Systems. E-mail: {rind, lammarsch, aigner, alsallakh, miksch}@ifs.tuwien.ac.at.*

Manuscript received 31 March 2013; accepted 1 August 2013; posted online 13 October 2013; mailed on 4 October 2013.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

1077-2626/13/$31.00 © 2013 IEEE Published by the IEEE Computer Society

data [2]. Yet, most available techniques deal with simply structured time-series data (time-value pairs). More complex temporal structures including intervals, multiple types of temporal primitives, temporal indeterminacies, multiple granularities, multiple calendars, multiple perspectives, and branching histories are only sporadically supported by current methods. Such methods are rather specific to their application domains and tailored towards a particular model of time.

There is a lack of software libraries that support the modeling, visualization, and processing of time-oriented data at a higher level than time-series data. Considering the importance of time-oriented data in many fields, more general support is desirable in order to ease the creation of new Visual Analytics (VA) methods and to increase their reusability. The need for an appropriate infrastructure has also been emphasized by Elvins [19] as well as by Silva and Catarci [46]. Moreover, building and providing reusable infrastructures is an important challenge called for in most VA research agendas [35, 48] and is seen as key for the growth of the discipline. Designing such an infrastructure is a formidable research challenge, not to mention the effort required to actually implement the library's broad functionalities. However, the gain from having such infrastructures available is significant [35, Ch. 6]. First, the usage of "standardized" components increases the quality of software compared to ad-hoc solutions. Second, results may be attained faster since the (re)implementation of basic infrastructural components is avoided. Third, both compatibility and comparability are increased if the solutions are based on common infrastructure. Such an infrastructure for VA of time-oriented data must be based on an expressive data model in order to capture the characteristics of time on a generalized level.

We propose the design of a versatile and reusable library that allows for easy modeling, visualization, and processing of time-oriented data, called *TimeBench*. After discussing the unique characteristics of time-oriented data (Section 2), the design requirements (Section 3), and related work (Section 4), we elaborate on the contributions of this work:

- A conceptual model of time-oriented data that describes time primitives explicitly and can express complex temporal data in a uniform data structure (Section 5).
- A software library that implements this conceptual model specific to the requirements of VA. It includes instants, intervals, spans, multiple granularities, as well as multiple calendars, and temporal indeterminacy. The library is based on established software design patterns, which make it easy to use by developers and are runtime efficient (Section 6).

We present three application examples built using this library in Section 8.1. These examples demonstrate both the expressiveness of TimeBench and the flexibility it offers for designing VA solutions for time-oriented data. In addition, three long-time developer studies in Section 8.2 report on its usefulness for research and student projects. Finally, Section 9 discusses how TimeBench fulfills its requirements, what limitations it has, and outlines possibilities for future work.

2 Time-oriented Data

Time-oriented data has an inherent structure that encompasses several aspects stemming from natural or social sources. Due to the importance of time-oriented data, its structure has been studied in numerous scientific publications (e.g., [2, 11, 23, 34]). As proposed by Aigner et al. [2], we divide the aspects of time-oriented data into general aspects required to adequately model the time domain and human-made abstractions that are useful to deal with the complexity of time-oriented data. The general aspects are *scale*, *scope*, *arrangement*, and *viewpoints*. In TimeBench, we focus on explicitly modeling the human-made abstractions, which we describe in the following:

Granularities. Time can be divided according to units that were originally derived from calendric systems. Analyzing data at the scale of several calendar units is important, e.g., when not only local artifacts affect the data but seasonal or weekly cycles as well. Also, when measurements are taken irregularly, it might be important to decide at which time points "steps" make sense. A full and formal definition of calendar units is given by Bettini et al. [11]. They base their work on a view on the discrete time domain that is composed of atomic units called *chronons*, which they represent by integer numbers. A *granularity* is defined as a mapping from chronons of the discrete time domain to subsets of these chronons. The authors also define a *granule* as a subset of chronons mapped by a certain granularity. Furthermore, they define grouping operations that allow for finer granularities to be grouped into coarser granularities. As example, if the chronons are days, they can be grouped into months or into years. Months can also be grouped into years. In actual calendars, the grouping operation usually happens periodically: all chronons in any January belong to the granule January. To define a particular January, the granularities month and year have to be combined.

Time Primitives. Several time primitives are defined by Goralwalla et al. [27]: *Instants* are a model for single points in time, *intervals* for ranges between two points in time, and *spans* are durations (of intervals) without a fixed position. Instants and intervals are anchored primitives, as they have a fixed position in time. Spans are unanchored primitives, as they do not contain information about their position. Allen [4] presents a set of 13 qualitative relations between two intervals as an extension of order theory to the time domain. This set can be expanded with further relations between instants and intervals [2]. An example for data where multiple time primitives are of importance are electronic health records, where medical tests are more or less instantaneous snapshots of a patient's state while conditions and therapies are present over longer intervals.

Determinacy. Time-oriented data can be subject to uncertainty in the time domain. Indeterminacy might stem from incomplete or inexact data in the application domain or from conversion between granularities (e.g., when weekly data are combined with daily data). Determinacy also plays an important role in project planning (e.g., when the duration of tasks is not fully known in advance) and medical treatment plans (e.g., latest possible beginning of a therapy) [2, 36].

The human-made abstractions of time-oriented data – granularities, time primitives, and determinacy – are related to each other and to the general aspects as well. Granularities can express a cyclic arrangement of time. Indeterminacy can be modeled by the time primitives *indeterminate instant* and *indeterminate interval*, which can be composed of standard intervals and spans [2, 3]. The time references of primitives are not necessarily chronons, but can be granules of any granularity. Thus, an instant or the begin of an interval can be anchored on a day granule. A span can be given as a number of granules and its length needs not to be a fixed number of chronons (e.g., a span of two months). Anchored time primitives as well as particular granules can be defined by their first and last chronons. These chronons are called the *infimum* and the *supremum*, which can be seen as functions $inf(x)$ and $sup(x)$ from primitives or granules to chronons. Figure 2 illustrates the interplay between granules and anchored primitives.

In the following section, we explain the requirements that arise when modeling the human-made abstractions of time-oriented data.

3 Desiderata

Based on the theory of time-oriented data (Section 2), existing work in VA of time-oriented data [2], and our own experience from multiple design studies, we establish the following desiderata that a VA library for time-oriented data should fulfill:

Expressiveness. The library needs to tackle the complexity of time-oriented data and support various aspects of this data such as primitives and granularities (Section 2). It must be flexible and extensible for a wide range of usage scenarios.

Common Data Structure. The library must offer standardized data structures as a common basis to allow recombining different visualizations, interactions, and automated methods in a polylithic fashion (cp. [8]).

Developer Accessibility. The library has to be as simple and slim as possible to ensure ease of use by software developers. The underlying data structures and operations should be exposed to the

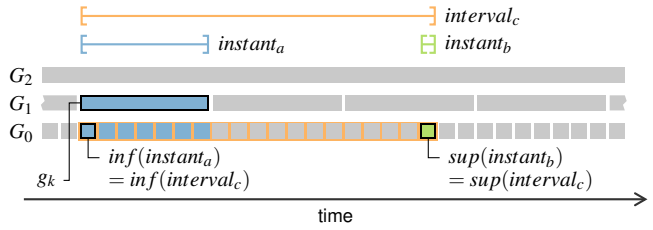

Fig. 2. Granules and anchored primitives. Granules of three granularities G_0, G_1, and G_2 (e.g., days, weeks, and months) are displayed as rectangles. Three time primitives are anchored to these granules. An interval $interval_c$ (orange) is defined beginning at $instant_a$ (blue) and ending at $instant_b$ (green). The first instant $instant_a$ is anchored at granule g_k of granularity G_1, which is composed of the seven blue chronons in the discrete time domain G_0. The second instant $instant_b$ is anchored at the green chronon in G_0, which implies that the ending of $interval_c$ is more precise than its beginning. The left-most blue chronon is the infimum of $interval_c$ and the green chronon is its supremum.

developer using an application programming interface (API) that "speaks the language" of the time domain. The library should be modular and easy to extend.

Runtime Efficiency. The data structures and the operations need to be optimized for interactive exploration in time-related scenarios. For this purpose, they have to take advantage of the different aspects of time-oriented data.

These desiderata compete with each other. For example, a high degree of expressiveness limits the options to optimize efficiency and leads to a more complex and less accessible architecture. As we discuss in Section 9, we prioritize the desiderata in TimeBench in the order as listed above, because TimeBench aims to support the rapid development of research prototypes.

We show in the next sections how TimeBench provides solutions to these desiderata and, thus, contributes to the vision of VA, beyond the state of the art.

4 Related Work

VA of time-oriented data is a wide area, which is illustrated by the over one hundred visualization techniques recently collected by Aigner et al. [2]. Some of these techniques are generic, like line plots, but many are design-study prototypes targeted at a specific domain problem. Moreover, these prototypes are built on diverse software infrastructures, which makes it hard to combine them in a multiple-view setting or to compare them in an evaluation framework.

Thus, related work can rather be identified in *software libraries* such as prefuse/Flare [32], ProtoVis [13, 31], D3 [14], Tulip [6], IVTK [20], Obvious [21], behaviorism [22], Simile TimeLine [33], and JFreeChart [25]. These libraries are focused on visualization and have limited or no support for automated methods. Applying them to build VA prototypes requires additional effort. In addition, there are *data analysis and visualization tools* such as Improvise [51], uVis [37, 38], Polaris/Tableau [47], SAS JMP, and MS Excel. These tools allow – to varying degrees – to interactively build VA prototypes. Some of them expose their APIs, which can be used as infrastructure for VA. An important criterion for software libraries and tool APIs is extensibility: While in some cases components can be customized only through a fixed set of properties (e.g., a bar chart widget), other libraries allow components to be refined by inheritance (e.g., a stacked bar chart widget as a subclass), or by composition (e.g., a stacked layout and a bar renderer as chart properties). The latter two possibilities are referred to as *monolithic* and *polylithic* architectures [8]. Next we analyze how the libraries and tools mentioned above support time-oriented data.

Time as Attribute. All the software libraries and tools are capable of working with data that has time points as one or more attributes. Internally, the time points are usually saved as numbers representing offsets in milliseconds, days, or years from some origin. Also dealing with input and output of various date/time formats is well supported. Many, but not all libraries and tools provide a linear time axis with major and minor tick marks based on calendar structures.

Granularities and Cycles. Calendar units are often used to group visual items and thus reveal temporal cycles. For example, the "reruns" view of Improvise [52] displays daily data as glyphs on a matrix, where the user can adapt how many glyphs fit in a row and, thus, change the cycle length. Weekend days are represented by a different glyph. Edges to separate the days by month and background gradients to represent seasons can be added in the background of the glyph matrix. Similarly, the "calendar view" demo of D3 [14] displays daily data in a pixel-based layout with week-of-year mapped to column and day-of-week mapped to row. This layout is achieved declaratively using date formats (e.g., "%w" for day-of-week) in the formulas that specify the coordinates. The tool uVis provides a Granularity component, which is used in different building blocks for time-oriented data [38]. For example, the CyclicalTimeAxis defines the x-axis mapping to build a Cycle Plot [16].

However, these operations work only with a fixed set of frequently needed granularities such as day-of-week but cannot be used in a generic way. Furthermore, they do not address that the granules of one granularity can be grouped to other granularities, which can be a source of temporal imprecision.

Time Primitives and Determinacy. While all libraries and tools can deal with data anchored to instants, other time primitives are less often supported. Interval data can be displayed as interval bars in the style of LifeLines [43] or Gantt charts in some of the tools, such as the "Gantt chart" component of Polaris/Tableau [47], the IntervalValues building block of uVis, and Simile TimeLine [33]. Nevertheless, there is a clear interest in such interval bar charts, which is illustrated by a variety of proposed workarounds to display a Gantt chart in MS Excel using stacked bar charts or error bars. The remaining time primitives are even less supported. We could not identify any library or tool that supported indeterminate intervals. Furthermore, the possibilities to mix different types of time primitives are limited.

Finally, VA is distinguished by the intertwinedness of visualization and automated methods, which are steered by coherent interaction methods, but these libraries and tools focus on the visual mapping of data and do not provide data transformations that go beyond that.

Due to the complexity and importance of time, there are also libraries that focus on dealing with time itself. τZaman [49] is a powerful client/server system that supports multiple calendric systems with multiple calendars and conversions between them. New calendric systems, calendars, and granularities can be specified in XML or by adding Java classes to define more complex structures. Joda Time [17] is a library that aims at replacing Java's default calendar classes. It supports multiple calendar systems and is focused on providing a simple API. Both libraries can be integrated as calendar backends in TimeBench, which is planned in future work.

5 Expressive Model for Time-oriented Data

Unlike the related libraries presented above, we propose a data model that describes time primitives explicitly. Thus, it is possible to work uniformly with different primitives (instants, intervals, and spans) in the same data structure. In addition, this allows for the extension with more complex time primitives such as indeterminate intervals, which are defined by an interval representing an imprecise begin, an interval representing an imprecise end, and two spans each representing the minimum and maximum of an imprecise duration [2, 3].

For this purpose, we introduce the *temporal dataset* as a generic data structure for time-oriented data. A temporal dataset $\mathscr{D}$ is the composition of temporal objects $\mathscr{O}$, temporal elements $\mathscr{E}$, and a timing function t from objects to elements:

$$\mathscr{D} = (\mathscr{O}, \mathscr{E}, t) \quad t : \mathscr{O} \to \mathscr{E} \tag{1}$$

The *temporal objects* $\mathscr{O}$ are a set of data items containing non-temporal attributes and are mapped to time by the function t. These attributes can be defined based on the requirements of the target domain. The model imposes no constraints on their number or data types and does not require a specific data structure; the temporal objects can be organized in a table, in a tree, or in a graph.

The *temporal elements* $\mathscr{E}$ are a set of time primitives, which hold the temporal attributes of the data, including information about the respective calendar granularities. As proposed by Lammarsch et al. [40], these can be instants that reference a granule or spans that reference a count of granules. Moreover, time primitives can form more complex structures specific to a target domain (e.g., indeterminate intervals). In order to express such complex primitives while keeping the attribute schema simple and consistent, time primitives can be built hierarchically. For example, an interval can be built from its begin instant and a span denoting its duration. In addition, a temporal element can be defined as a subset that groups temporal primitives without a specific structure. To improve the expressiveness even further, this hierarchy is organized as a directed acyclic graph, which allows a primitive to have multiple parts and be part of multiple other primitives.

The *timing function* t maps each temporal object to exactly one temporal element. Conversely, the inverse function t^{-1} maps each temporal element to a (possibly empty) set of temporal objects. This 1-to-n relationship between objects and elements is an adequate trade-off between expressiveness and accessibility. Without loss of generality, the temporal dataset can model objects having multiple temporal elements. For this purpose, multiple timing functions can be used, if a fixed number of temporal elements are needed for each object (e.g., an electronic health record with date of birth and period of treatment). To support an arbitrary count of temporal elements per object, these elements can be grouped under a composite time primitive (e.g., administration time of a medication). If an object without timing information is needed, it can be mapped to a special primitive that spans over the complete calendar or the lifespan of the dataset.

Example. We demonstrate our data model by illustrating some items of an electronic health record as depicted in Figure 3. Each entry contained in the record can be related either to a time point or to a time interval. Medical encounters, medical tests, or images are examples of instantaneous entries, which are represented by temporal objects that are mapped by the timing function to an instant:

$$o_1 = (Pulse, 92) \quad t(o_1) = i_1 = instant_{minute}(2013\text{-}02\text{-}21\,16{:}10) \quad (2)$$

An example of interval data is a health condition such as influenza from one medical encounter to another one. This temporal object is mapped to an interval that is built from two instants (begin, end) representing the time of the medical encounters. Another example is a drug prescription from today for seven days, which is mapped to an interval built from an instant and a span (begin, duration).

Next, we explain how software design patterns can be extended with this time-oriented data model in order to flexibly support a wide variety of time-oriented datasets.

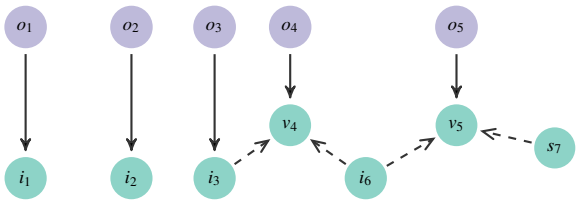

Fig. 3. Example of a temporal dataset on the conceptual level. The temporal objects o_1–o_3 represent data at the time points i_1–i_3 (e.g., blood tests), whereas o_4 and o_5 occurred during time intervals v_4 and v_5 (e.g., artificial ventilation). Dashed arrows denote the building of composite primitives such as interval v_4, which begins at i_3 and ends at i_6, and interval v_5 beginning at i_6 for the time span s_7.

6 Library Design

The following sections give an overview of the library's architectural design choices and abstractions. TimeBench focuses on supporting time-oriented data. For general visualization features it reuses software components from an existing polylithic visualization library, in our case, prefuse [32]. The software design of TimeBench is based on established *design patterns for Information Visualization* [30], which makes it possible to apply its abstractions in different software environments. The overall architecture follows the *Reference Model* pattern[1] and separates abstract data, visual data, views, and interaction techniques. A major part of TimeBench is the implementation of the expressive data model for time-oriented data in *data structures* that are accessible to developers and efficient to run. This extends the abstract data component of the *Reference Model* as we show in Section 6.1. Another important part is the *calendar* component (Section 6.2), which provides operations for calculation with granularities and granules. On this basis, various *data transformations* as well as *visual mappings, renderers, and interactive controls* for VA of time-oriented data are possible. Interaction passes throughout the architecture encompassing both visual mapping and data transformations, which allows for VA. We present some examples in Section 6.3 and Section 6.4 to demonstrate the general applicability of our design.

The diagrams in this section use the *extended object-modeling technique notation* [24, 30] to depict classes and their relations. A circle at the end of an arrow indicates a 1-to-n relationship and a diamond at base of a relation denotes aggregation. Classes with white background are part of TimeBench, whereas gray background denotes classes preexisting in the software environment (i.e., prefuse). The diagrams show an abstract view and do not specify all details of the implementations.

6.1 Data Structures

TimeBench realizes the data model proposed in Section 5 and allows developers to work with temporal objects and temporal primitives in an object-oriented fashion. It provides factory and accessor methods, time-specific indexing structures, and input/output features.

Internally, temporal objects and temporal elements are stored as rows in *relational data tables*. This data structure is efficient and flexible thanks to the *Data Column* pattern and can be integrated directly with relational databases. For convenient development in an object-oriented fashion, each table row can be accessed as a tuple object, which is backed by the data residing in the data columns (*Proxy Tuple* pattern, a variant of the *Facade* pattern). Graph or tree data structures can also be built using relational data tables by storing the nodes and the edges as tuples in separate tables (*Relational Graph* pattern).

Figure 4 gives an overview of the data structures in TimeBench. The central class is TemporalDataset, corresponding to $\mathscr{D}$ from the model. Tuples in this data set are TemporalObjects. Internally the temporal dataset is composed of a number of data columns holding non-temporal attributes, $\mathscr{O}$, and a TemporalColumn. This specialized data column holds one reference to a temporal element for each row, which realizes the timing function t. The TemporalElement tuples, $\mathscr{E}$, are created and stored by the TemporalElementStore.

By default, TemporalDataset is a subclass of Graph, which makes it compatible with existing visualization components. It holds temporal objects as graph nodes and stores their relationships as non-temporal, directed edges. Temporal objects have exactly one temporal column, with a predefined attribute name and accessor functions. However, this is just a common default structure, which we adopted based on various prototypes we created so far. TimeBench, just like the time-oriented data model, does not limit the number of timing functions nor how the data is organized. Temporal columns can be added to a flat table, to a tree, or to the edges of a graph. Additional temporal columns can be added to define additional timing functions.

[1] Design patterns are denoted in italics and refer to Information Visualization design patterns described by Heer and Agrawal [30] or general software design patterns described by Gamma et al. [24].

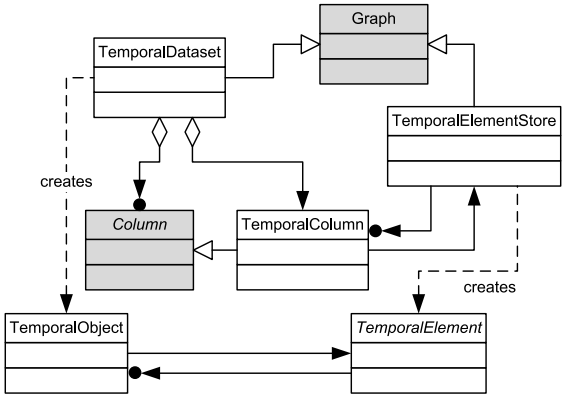

Fig. 4. Overview of the data structures. TemporalDataset and TemporalElementStore are subclasses of Graph, and allow access to TemporalObject and TemporalElement tuples respectively.

Table 1. Columns of the TemporalElement Table.

Column Name	Type	Explanation
id	long	unique identifier
inf	long	first chronon for anchored elements granule count for unanchored element
sup	long	last chronon for anchored elements granule count for unanchored element
granularityID	int	identifier of the granularity
granularityContextID	int	identifier of the context granularity
kind	int	enumeration of primitive types (0 = span, 1 = set/custom temporal element, 2 = instant, 3 = interval)

The TemporalElementStore is a separate data structure and can be shared between different temporal datasets. This has the advantage that the same temporal elements can point both to original data and to filtered or derived data. It is also possible to use one temporal element store throughout a scenario as a *Singleton* object. The TemporalElementStore references back to the TemporalColumns where it is used. This allows listing all temporal objects timed at a specific temporal element.

The TemporalElementStore has a uniform structure that is stable across all scenarios. The temporal elements are stored as nodes in a directed acyclic graph and have a fixed table schema (Table 1). Depending on whether a temporal element represents an anchored primitive or an unanchored primitive, the semantics of the "inf" and "sup" columns differ. For an anchored primitive these columns hold the first (infimum) and the last (supremum) chronon of the element. For an unanchored primitive they both hold the length in granules of the element. In addition, there are columns that denote the granularity, a switch for the primitive type, and a unique element identifier. The edges of the graph represent the hierarchical structure of composite primitives. They can also be interpreted as aggregation to sets or as custom relationships between temporal elements to allow for wider flexibility. Thus, TimeBench is optimized for performance in regard of the temporal occurrence, while retaining a common and expressive data structure.

Due to the different semantics of the columns, creating the temporal elements and manipulating their attributes and relationships directly can be cumbersome and prone to mistakes. Therefore, TimeBench instantiates *Proxy Tuple*s as objects of a specific subclass of TemporalElement (Figure 5): The classes Instant, Interval, and Span have dedicated accessor methods that handle the respective aspects of the time domain. Additionally, the TemporalElementStore provides dedicated *Factory Method*s for these primitives. These classes also interface with the calendar package, as they take Granule and Granularity objects as input and output. For example, an instant can be created from a granule, and an interval can be created from an instant (begin or end) and a span (duration). Custom primitives can be defined as subclasses that implement their particular semantics in an object-oriented fashion. In addition, the subclass GenericTemporalElement allows direct, unrestricted access to the internal data. If needed, developers can switch seamlessly between these two perspectives because these objects save their attribute values not locally but in the TemporalElementStore.

The availability of *factory and accessor methods* allows maintaining good-practice rules for dealing with time-oriented data. For example, anchored primitives should begin with the infimum (first chronon) of a granule in their granularity. Yet, TimeBench does not check or enforce such practice in order not to limit flexibility and efficiency.

The efficiency of data lookup is improved further through *index structures*. In particular, the interval index [18] allows efficient access to temporal objects by their occurrence time. Specifically, it supports the full range of qualitative relations [4] between the queried temporal objects and a given time point or interval. For example, it is possible to look up temporal objects that occur within or intersect with a time interval. These relations can be implemented on top of the IntervalComparator interface. The interval index is implemented as a red-black tree, which guarantees $O(\log n)$ algorithmic complexity both for adding and removing elements to the index, and for performing the temporal queries. TimeBench updates the index automatically upon changes in the temporal elements.

TimeBench provides a range of *input/output* features. There are importers for calendar data in iCal format and comma-separated text files in various data layouts. If the layout and the format of temporal data are not automatically detected, it can be specified in a metadata file. Data exchange between TimeBench and the R Project for Statistical Computing works bidirectional, supporting the R classes ts and zoo. TimeBench data can be saved as a GraphML document that retains the internal structure of the TemporalDataset as closely as possible.

6.2 Calendar Operations

The classes of the calendar package offer an intuitive interface to developers hiding the complexity of the powerful calendar operations described by Bettini et al. [11] and Goralwalla et al. [26]. These operations mostly revolve around Granule objects (see Section 2) and integer arithmetic with identifiers and inf/sup values. Typical operations are (1) checking the qualitative temporal relations [4] between granules, e.g., whether they overlap or which is before the other; (2) shifting granules by a given number of identifiers (e.g., two months) or chronons using robust methods for chronons that do not fit a granule; (3) the conversion of granules on one granularity to the corresponding granules on another granularity (e.g., hours to days). This operation can determine the extent to which a granule on one granularity lies within another granule on a different granularity. An exam-

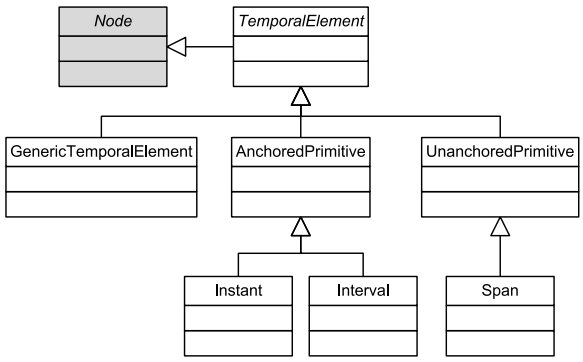

Fig. 5. Subclass hierarchy of temporal elements. Different temporal primitives are first-class objects.

ple application for this is the organization of temporal objects in tree structures based on granularities (see GranularityAggregationAction in Section 6.3).

Dealing with calendars is mainly related to the Granule class. Table 2 lists its attributes. A granule is defined by its identifier, its granularity, and a context granule. For example, a "January" granule might have the number 0 as identifier and the granularity "month in context of year". Depending on whether the contextGranule is specified or not, the granule is either particular or general. For example, a general granule is just January, without further specification. A particular granule is the January of 1987. General granules can be used to model temporal cycles (every January) or indeterminacies (some January). In both cases, there are as many identifiers as there are different granules on their granularity. E.g., for the granularity month of year, the identifier can range from 0 (January) to 11 (December).

Like an instant or an interval, a particular granule has an inf value and a sup value representing its first and last chronons. Thus, the context granule also determines whether the granule has a temporal location or not. A general granule, having no context granule, does not need to have values set for inf and sup. Furthermore, each granule is given a human readable label that depends on the localization of the environment.

To create a Granule on a given Granularity (see below for granularities), users have to either provide the identifier or the chronons inf and sup. The other values are calculated on-demand. TimeBench provides a number of variants for dealing with fuzzy input: If the provided inf or sup do not correspond to the inf and sup of any granule in the provided granularity, it is possible to use the granule around the inf, the granule around the sup, or the granule around their mean. It is also possible to create an array of Granule instances that fill a certain interval of time between two given chronons. Granule objects can also be directly created from temporal primitives, such as Instant and vice versa. In this case, the granule object is cached by the TemporalDataset so that related operations, such as the calculation of an identifier do not need to be performed multiple times upon subsequent queries for the same granule. To reduce the computational overhead further, granule objects are created on-demand only. Also, when exporting the data, granules are not exported with it, since they can be recalculated automatically upon import.

As stated above, each granule is dependent on a granularity. Without a granularity, operations like creating identifiers or checking whether the inf and sup fit in the boundaries of an actual granule are undefined. Thus, TimeBench also provides a Granularity class. An instance of Granularity is required to create a Granule instance. Each granularity has its own identifier (e.g., day) as well as a context identifier (e.g., week or month). Granularities in turn might have different meanings depending on the calendar they belong to. Therefore, like a Granularity instance is needed to create a Granule instance, to create a Granularity instance developers need an instance of the Calendar class. Each Calendar instance has a unique identifier as multiple calendars might be needed in the same application.

Several different implementations of calendric systems are possible, each supporting a different set of calendars that all might be needed by

Table 2. Attributes of the Granule Class.

Attribute	Explanation
inf	The infimum, the first chronon of the granule.
sup	The supremum, the last chronon of the granule.
identifier	The granule identifier, its global number or its number inside the context granule.
label	A human readable label for the granule (e.g., a 0 for identifier can become "January").
granularity	The granularity to which the granule belongs.
contextGranule	The context granule. If this it null, the granule will be a general one (e.g., a general January).

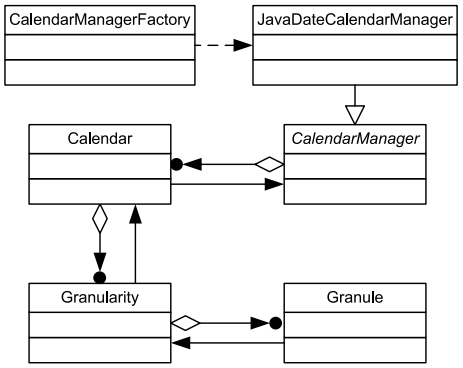

Fig. 6. The classes of the calendar package support multiple granularities and multiple calendars. A calendar manager implementation is provided using the *Strategy* pattern.

developers, or, respectively, their users. Some very powerful systems, like τZaman [49], are very flexible in defining and handling custom calendars (e.g., Academic) and granularities (e.g., business week), but are rather complex to set up and connect to. Therefore, we made the design decision to implement granules, granularities, and calendars in a generic way as *Facade* to different calendar backends. The calendar package is as flexible and extensible as its backends. It delegates the calendar calculations to classes that implement the CalendarManager interface (Figure 6). This structure of the calendar package allows integrating different calendar backends with minimal need for changes to existing code (*Strategy* pattern). The runtime performance largely depends on the calendar manager and its backend. So far, we have implemented the JavaDateCalendarManager, which performs all calculations based on the methods provided by the Java core classes and supports the Gregorian calendar only.

To access different calendar managers at runtime (e.g., when loading a configuration file) an identifier is needed to retrieve them from the CalendarManagerFactory class. Next, developers can create an instance of the Calendar class by specifying a calendar manager and the identifier of the required calendar.

Finally, the calendar package also is our way of modeling relative time. It is possible to create a calendar with a given origin (if supported by the calendar manager). Commonly used origins are the birth of Christ or of UNIX. By specifying a manual origin, all granule identifier calculations are performed according to this origin, while the chronon level remains unchanged.

6.3 Transformations on Data Tables

In TimeBench, automated analysis methods can be implemented as data transformations over the TimeBench data structures described in Section 6.1. By means of an extension we introduce to the software library prefuse, user interactions can not only modify the visual mapping but also perform data transformations. These transformations can be implemented in the same way as in prefuse without being limited to visual attributes. Two design patterns can be used for this purpose: The *Operator* pattern, which is implemented as Action in prefuse, can be used to transform a table or a graph into another table or graph (e.g., a TemporalDataset into another), or to compute additional values in a procedural fashion (e.g., results from visual mappings). The *Expression* pattern can be used to define the transformation in a more functional way of programming. Existing actions or expressions from the prefuse packages can be seamlessly applied to TimeBench data structures and be combined with TimeBench data transformations. For example, a pattern finding method can use the built-in ComparisonPredicate to handle non-temporal attributes and TimeBench expressions for temporal attributes.

One novel action provided in TimeBench to support various automated analysis methods and visualizations is the GranularityAggregationAction. This action arranges the temporal objects as leaves of a tree that is generated based on a set of granularities. The branches are

filled with aggregated values for the non-temporal data, allowing for various aggregation methods, like mean or sum. For example, hourly values can be aggregated to daily means and subsequently to yearly means. The generation of the tree is supported by the calendar package (Section 6.2). Using the GranularityAggregationAction, visualization methods, like Cycle Plot [16] or GROOVE [39] have already been implemented in TimeBench (see Section 8.1).

We have also implemented time-oriented expressions that represent a temporal element or an array of temporal elements. Based on these expressions, we have implemented predicates that evaluate if the primitive type, granularity, or context granularity of these temporal elements matches the query. There are also expressions for shifting temporal elements by a given amount of chronons. Finally, the TemporalComparisonPredicate evaluates whether two expressions representing two temporal elements or a temporal element and a literal anchored primitive are in a specified qualitative temporal relation (e.g., before, starts, overlaps) [4]. These expressions can also be combined. For example, by shifting a temporal element a certain number of chronons and testing for the *meets* relation, it is possible to check whether it ends exactly a given time before a given instant. These expressions can be used both to filter and aggregate the data and to define the visual mapping in existing actions, such as conditional coloring.

6.4 Visual Mapping, Rendering, and Interaction

Since the TemporalDataset is a subclass of the built-in Graph data structure, it can be added to a regular prefuse Visualization object. Prefuse creates a VisualGraph that inherits all data columns from the Graph and adds columns for visual attributes (*Cascaded Table* pattern). Thus, non-temporal attributes of the temporal dataset are immediately available for prefuse visual mappings such as AxisLayout. Likewise, all built-in renderers and interactive controls in prefuse are compatible with the new data structure.

Leveraging temporal attributes in the visualization is possible in multiple ways thanks to the polylithic architecture of TimeBench and prefuse. First, there are standard components available in TimeBench such as TimeAxisLayout. Laying out temporal objects along a linear time axis is possibly the most common visual mapping of time. For this purpose, the TimeAxisLayout comes with a suite of interactive zooming and panning controls and gridlines. The gridlines and labels use calendar granules for meaningful time units and expand or collapse automatically during interactive zoom. Second, the developer can implement custom visual mappings, renderers, or interactive controls that are aware of the TimeBench data structures. Prefuse supports this customization by means of several base classes that can be extended for easily achieving the desired visualizations. Third, the developer can use the *Expression*s and *Operator*s presented in Section 6.3. This can, e.g., be used to color visual items according to the granule they belong to, whether they happen during a certain range of time, or according to any other qualitative temporal relation.

Alternatively it is possible to add the TemporalElementStore to the Visualization as a VisualGraph. However, in this case custom components are needed to access the actual data which is stored in the TemporalDataset.

7 Implementation

In Section 6 we elaborated on the software architecture and abstraction for a library to support VA of time-oriented data. In order to allow widespread use of the library by the research community as well as by practitioners, we make it available as open-source software under a BSD 2-Clause license. Its implementation and instructive demos are available at `http://www.timebench.org`. Furthermore, we continue extending the library at GitHub (`https://github.com/ieg-vienna/TimeBench`) and we invite the community to contribute via this platform.

TimeBench is implemented in the Java programming language, version 1.6, and uses the polylithic visualization library prefuse [32]. In addition it depends on Apache Commons Lang 3.0, Apache log4j 1.2, iCal4j 1.0.4, and on the Java/R Interface (JRI), which is part of rJava. Currently, TimeBench is comprised of 136 classes with about 13,000 lines of code.

8 Evaluation

We investigate the applicability and usefulness of TimeBench using application examples and long-term developer studies. The development of VA solutions cannot be broken down to routine activities. Therefore, we apply qualitative evaluation methods, which offer more realistic results.

8.1 Application Examples

The following examples were chosen to demonstrate TimeBench under different data characteristics and challenges (Table 3). Their implementations by the authors are available from the project webpage (`http://www.timebench.org`).

Horizon Graph. In the first example (Figure 1.1), we compare multiple time-series over a linear time axis using the Horizon Graph technique [44]. For the visual mapping of time to the x-coordinate, we use the TimeBench component TimeAxisLayout (see Section 6.4). This layout provides mouse controls together with user interface elements that allow the user to pan and zoom in time. The time axis labels automatically expand or collapse accordingly based on calendar units. As data transformation, indexing is often used together with horizon graphs and is an effective technique to make time series with largely different value ranges visually comparable [1, 9]. For this example we use the TimeBench component IndexingAction with a mouse control that interactively sets the indexing point to the temporal element closest to the point clicked with the mouse. This shows the development relative to that point in time. For the y-coordinates, colors, and areas we do not need to tackle temporal aspects.

GROOVE. This example (Figure 1.2) comprises an implementation of Granularity Overview OVErlay (GROOVE) visualizations, a pixel-based visualization technique specialized for time-oriented data [39]. It requires that the temporal objects are organized in a calendar-based tree structure that has input values in the leaves and aggregated values in their parents, up to the root. We create such a structure from a table dataset using the GranularityAggregationAction provided by TimeBench (see Section 6.3). The temporal objects of various levels in the tree are depicted as nested rectangles, whose positions are determined based on the structure of the tree and on the identifiers of granules created from their temporal elements. This placement is performed by a layout action we developed for GROOVE, but would also be suited for the time axis of a Cycle Plot [16]. In a pixel-based visualization, numerical values are mapped to color. In our example, we show a color overlay variant of GROOVE. This variant maps the values of two aggregation levels to hue and lightness [39]. We have implemented a specific OverlayDataColorAction for this purpose.

PlanningLines. Indeterminacy is the main challenge of the third example (Figure 1.3) where a project schedule or a medical treatment plan is visualized. We use the visual metaphor of PlanningLines [3],

Table 3. Characteristics of the Application Examples.

Example	Data/Time	Challenges
Horizon Graph	instants, numerical, multivariate	linear time axis, interactive indexing
GROOVE	instants, numerical, univariate transformed to tree with 5,965 nodes	granularity-based aggregation & layout, color based on two granularities
PlanningLines	indeterm. intervals, nominal, graph	indeterminacy, scheduling

which is similar to a Gantt chart [28], to represent tasks that are scheduled during an indeterminate interval. The intervals between earliest and latest begin as well as earliest and latest end are shown as black caps, while the minimum and maximum duration are shown as colored bars. We use the hierarchical structure of temporal elements to store these complex primitives and we apply the decorator item feature of prefuse to render the individual parts of the PlanningLines metaphor. For the encoding of time, we reuse the linear time axis component of TimeBench, in particular the IntervalAxisLayout. The dependencies between the tasks are stored as directed graph edges and are represented as arrows.

Summary. Even though each example would be possible without TimeBench, we observed that the code complexity did not increase by using TimeBench while the code volume decreased through reusable components such as the time axis and granularity aggregation. In particular, GROOVE depends heavily on calendar operations and hierarchical structure of temporal objects. PlanningLines use indeterminacy to demonstrate the need for complex temporal primitives.

8.2 Long-term Developer Studies

In addition to the application examples which re-implemented existing work within a time frame of hours or days, TimeBench was applied in research and student projects for a period of several months. Next, we report observations on its usefulness for some of these projects.

TiMoVA. This project was conducted by one M.Sc. student as part of his master thesis in computer science. The student had no previous experience with TimeBench or prefuse. As a result of the work a VA solution for the selection of time series models in a highly interactive visual interface has been designed, implemented, and evaluated [12].

The prototype loads time series data from comma-separated text files into a TemporalDataset. It uses the bidirectional interface to R because it performs statistical computations in R and shows diagnostic plots using TimeBench/prefuse. It was relatively easy to implement direct manipulation of model parameters within the plots. For visualizing the time series and the residuals over time, TiMoVA provides line plots built using the time axis component of TimeBench.

Summary. The student could adapt the line plots based on existing demo code. He could reuse existing data import/export interfaces and TimeAxisLayout. Therefore, he did not need to learn how TimeBench works internally or re-implement standard features but he could focus on statistical methods and on designing a smooth user experience.

DOI Time Scale. The second developer study reports on the final project of two high-school students specialized in information technology (19 years old). The students had gained some experience in TimeBench and prefuse during a one-month internship with the authors. Before that, they had only developed in C#. The students developed extensions of the linear time axis component of TimeBench addressing two issues of visualizing irregularly sampled data [7].

The first extension tackles large gaps in the data by compressing the time scale at gaps larger than a user-defined threshold. The second extension allows the user to distort the time axis in a way similar to a fisheye lens. For this the GranularityAggregationAction is used to compute aggregated values.

Summary. The students did not only reuse TimeBench components but also extended some of them. They recorded a total development time of 165 person hours (ph): 10 ph for setup on basis of a demo, 46 ph for the gap time scale, and 109 ph for the DOI time scale.

Temporal Pattern Discovery. Finally, we report on a research project conducted by four TimeBench authors and one external expert in knowledge discovery in databases. The outcomes are a new pattern finding approach that extends existing work [10] to better capture the temporal aspects in event data and an interactive visualization for exploring the resulting patterns in such data [41].

The pattern finding approach first searches for events, which are defined as consecutive temporal objects that fulfill specific conditions (e.g., value above the threshold for all objects). The conditions may contain temporal aspects (e.g., event duration is more than one hour or includes a Monday). Then, it iteratively chains events to patterns, whereby one of a number of temporal relations between events must be fulfilled (e.g., meets). The implementation uses separate TemporalDatasets for input data, events, and patterns. The patterns are stored as a forest with the temporal relations represented as tree edges. Each path from a root to a leaf represents a pattern instance. Both steps are implemented as data transformations and they use temporal predicates to specify the event-forming conditions and the temporal relations between the events (Section 6.3).

The visualization shows the patterns as an Arc Diagram [50] with the events depicted as colored bars in the middle of the view. In the example in Figure 1.4, there are three colors representing three categories of events. The arc color depends on the category of the first event in the respective pattern.

Summary. TimeBench supported the development in multiple ways: (1) The data structures, in particular storing patterns as a forest, help in avoiding difficulties which some team members encountered while implementing a predecessor approach [10]. (2) The two steps of the pattern finding approach are implemented as separate *Operator*s, which helps in improving code readability and reusability. (3) The temporal predicates provide a high flexibility, making not only new parameters, but also completely new kinds of parametrization possible without changing the algorithm itself. (4) All information encompassing the input data, the events as intermediate results, and the patterns as final results can be added to the same visualization, because TemporalDataset is compatible with prefuse data structures. (5) The Arc Diagram relies largely on existing TimeBench/prefuse components (e.g., linear time axis). The only module that had to be developed was the ArcRenderer.

9 Discussion

In this section, we demonstrate how TimeBench can fulfill the desiderata from Section 3 based on the application examples and developer studies presented in Section 8.

Expressiveness. Our expressive model for time-oriented data described in Section 5 supports time primitives as well as determinacy by building more complex time primitives from simple ones. Temporal datasets can be as simple as a time series, or more complex with a mixture of various time primitives as the one shown in Figure 3.

The calendar package introduces granularities in a way that allows for developing complex calendar managers while working with well-known human abstractions only. This includes full support for the frequent tree structures in calendric time as well as cycles, like seasons.

We have validated the expressiveness of TimeBench by using it in several research projects such as Temporal Pattern Discovery (Section 8.2). In addition, Section 8.1 presents three examples: The Horizon Graph shows multivariate time-series data, the GROOVE visualization shows several levels of granularities at the same time, and PlanningLines show the handling of indeterminate intervals.

Common data structure. We have presented a data structure in Section 6.1 that is suitable for all VA scenarios dealing with time-oriented data that we could conceive. In Section 8, we present application examples and projects that demonstrate various kinds of time-oriented data and various kinds of suitable VA methods that are all using this data structure. Internally, data is stored as rows in relational data tables. This data structure is efficient and flexible and can be integrated seamlessly with relational databases to improve scalability.

Developer accessibility. We differentiate between developers who use or recombine existing TimeBench components and developers who implement new or derivative components. For the first group, the demo programs provided with the library serve as templates and as source for copying code snippets, which is a frequently used approach to familiarize oneself with a new API [32]. The TiMoVA project serves as an example for this. Thus, it is possible to build visualizations such as a line plot or a Gantt chart very quickly without the need to understand the inner workings of TimeBench. In fact, the main difference between a prefuse scatter plot and a TimeBench Gantt chart is the change of AxisLayout to IntervalAxisLayout. Without TimeBench the developer would need to implement a data structure for intervals and a layout that considers intervals themselves.

The second group of developers needs to understand the design of TimeBench. To facilitate this, TimeBench is built on software design patterns and uses common terminology for time-oriented data. Furthermore, TimeBench provides an object-oriented API that uses the *Proxy Tuple* pattern to access its internal data structure and the *Facade* pattern to encapsulate complex calendar logic. Modeling the non-temporal data aspects works as with prefuse. We have evaluated the accessibility of TimeBench with two high school students who extended TimeBench as part of their final project. They experienced a steep learning curve, but once they mastered prefuse and how it uses software design patterns they adopted TimeBench relatively quickly and finalized their project with great success. Another obstacle was the incomplete documentation, which was due to the project being performed parallel to finishing TimeBench.

Runtime Efficiency. The efficiency can be appraised using our application demos. They run smoothly and responsively on standard hardware. The Horizon Graph shows 3,360 temporal objects. In the GROOVE demo 5,115 original and 850 aggregated objects are visible. The Arc Diagram presents 3,842 different patterns.

To increase efficiency, we would have to sacrifice expressiveness or accessibility. In this trade-off we focus on the latter two, as TimeBench is a framework designed mainly for developing research prototypes.

Limitations. The flexibility of TimeBench stems from more complex data structures which impose an overhead in memory requirements. However, a linear overhead is usually not a severe problem on modern computer hardware. The main recipe to limit memory consumption is using efficient algorithms.

TimeBench has been developed in Java like many existing research libraries. As a result, web-based applications can only be deployed as Java applets or by Java Web Start, which do not seamlessly integrate with web pages and are often disabled for stability and security reasons. However, our concepts and the design patterns we use do not rely on Java and can be replicated in other languages.

The JavaDateCalendarManager only provides a Gregorian calendar which does not allow specifying a user-defined origin but always centers around 1970-01-01 0:00:00,000am. The calendar package in general does not have this limitation, thus this implementation detail has to be addressed by creating a more versatile calendar manager.

Future Work. As an open, polylithic library, TimeBench can be extended in many ways, such as introducing additional actions for visual mapping and automated methods or designing user interactions.

We do not currently support huge datasets or streaming data. However, integrating the in-memory data structures of TimeBench with relational databases, big data repositories, or data streams would be possible. In this case, only parts of the data are loaded at a time, to generate smaller aggregated temporal datasets from them that are subsequently used. A possible way to achieve this is to support the interoperability with the meta library Obvious [21], which was designed to facilitate such data integration efforts.

The calendar package allows integrating new calendar managers. The current JavaDataCalendarManager has been rapidly developed to allow using the Gregorian calendar. As our final goal is to provide more powerful calendar support, we are in the early stages of developing a calendar manager that connects to the τZaman system [49]. This calendar manager should also support the relative time model.

Another aspect of time-oriented data that we do not directly model in TimeBench is branching time [2]. The tree structures we make available for temporal objects as well as for temporal elements allow the developers to model branching time manually. However, it would be more expressive to extend the TimeBench data model to handle this kind of time-oriented data.

Finally, time is not the only dimension that has an inherent structure. Other types of data, like spatial data, can be very complex and much research has been put into understanding their structures. Based on our data model, it is worth to investigate the integration of spatial elements to model spatial data, in a similar fashion as our temporal elements.

10 Conclusion

Time-oriented data plays an essential role in many VA scenarios. However, there is no reusable software infrastructure that provides foundational data structures and algorithms for time-oriented data in VA. Such infrastructure should respect that the time domain and time-oriented data have a complex structure. It should facilitate reuse through a common data structure, provide an accessible API to developers using concepts from the time domain, and provide adequate runtime performance.

In creating the software library TimeBench, we fill the gap by providing this infrastructure. In this paper we presented its underlying conceptual data model and elaborated on its architectural design and abstractions. In addition, we provide the library as open-source software. From designing this library, we can deduct the following lessons learned:

- The complexity of time-oriented data (granularities, time primitives, and determinacy) can be expressed in a common data structure by modeling time primitives explicitly using a simple and consistent table schema and hierarchical composition.
- By applying the *Proxy Tuple* pattern to enhance data tuples with a time-specific API, we make it easy for developers to build visual mappings, interactive controls, and data transformations for time-oriented data.
- Powerful calendar operations based on granularities provide VA techniques with a solid basis for insight discovery. The calendar manager encapsulates these operations, which is advantageous when connecting to more complex calendar managers and when dealing with relative and branching time.
- By explicitly modeling the structural aspects of time, data transformations such as granularity aggregation and qualitative temporal relations can directly access these aspects and be implemented in a much simpler fashion than based on "raw" data. This also holds for visual mappings, like mapping granule identifiers directly to locations on various axes, or mapping the infimum (first chronon) and supremum (last chronon) of an interval to two locations on the same axis.

We demonstrated TimeBench by means of three application examples, reported on three projects built on top of it, and discussed how it fulfills its design requirements. Thus we can conclude that (1) it eases the development and testing of new visualization, interaction, and analysis methods for time-oriented data; (2) it facilitates the reuse and the combination of such methods thanks to its common data structure; and (3) thus, it fosters the reproducibility and comparability of VA techniques for time-oriented data. Consequently, we assume that TimeBench is a valuable asset when developing research prototypes for VA of such data and is extendable to address future challenges in this area.

Acknowledgments

This work was supported by the Austrian Federal Ministry of Economy, Family and Youth via CVAST, a Laura Bassi Centre of Excellence (#822746), and by the Austrian Science Fund (FWF) via the HypoVis project (#P22883). Many thanks to Peter Weishapl, Michael Atanasov, Philipp Schindler, David Bauer, Sascha Plessberger, and Markus Bögl for their implementation contributions.

References

[1] W. Aigner, C. Kainz, R. Ma, and S. Miksch. Bertin was right: An empirical evaluation of indexing to compare multivariate time-series data using line plots. *Computer Graphics Forum*, 30(1):215–228, 2011.

[2] W. Aigner, S. Miksch, H. Schumann, and C. Tominski. *Visualization of Time-Oriented Data*. Springer, London, 2011.

[3] W. Aigner, S. Miksch, B. Thurnher, and S. Biffl. PlanningLines: Novel glyphs for representing temporal uncertainties and their evaluation. In *Proc. Int. Conf. Information Visualisation (IV)*, pages 457–463, 2005.

[4] J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[5] N. Andrienko and G. Andrienko. *Exploratory Analysis of Spatial and Temporal Data: A Systematic Approach*. Springer, Berlin, 2006.

[6] D. Auber. Tulip – a huge graph visualization framework. In M. Jünger and P. Mutzel, editors, *Graph Drawing Software*, pages 105–126. Springer, Berlin, 2004.

[7] D. Bauer and S. Pleßberger. *DOI TimeScale for Medical Histories*. Diplomarbeit, HTBL Krems, Supervisors R. Wenzina and W. Aigner, 2013.

[8] B. B. Bederson, J. Grosjean, and J. Meyer. Toolkit design for interactive structured graphics. *IEEE Trans. Software Engineering*, 30(8):535–546, 2004.

[9] J. Bertin. *Semiology of Graphics: Diagrams Networks Maps*. University of Wisconsin, Madison, 1983. Originally published in 1967 in French.

[10] A. Bertone, T. Lammarsch, T. Turic, W. Aigner, S. Miksch, and J. Gärtner. MuTIny: A multi-time interval pattern discovery approach to preserve the temporal information in between. In A. P. dos Reis and A. P. Abraham, editors, *Proc. IADIS European Conf. Data Mining*, pages 101–108, 2010.

[11] C. Bettini, S. Jajodia, and S. X. Wang. *Time Granularities in Databases, Data Mining, and Temporal Reasoning*. Springer, Berlin, 2000.

[12] M. Bögl, W. Aigner, P. Filzmoser, T. Lammarsch, S. Miksch, and A. Rind. Visual analytics for model selection in time series analysis. *IEEE Trans. Visualization and Computer Graphics*, 19(12), 2013. forthcoming.

[13] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE Trans. Visualization and Computer Graphics*, 15(6):1121–1128, 2009.

[14] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Trans. Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[15] S. Card and J. Mackinlay. The structure of the information visualization design space. In *Proc. of IEEE Symp. on Information Visualization (InfoVis)*, pages 92–99, 1997.

[16] W. Cleveland. *Visualizing Data*. Hobart Press, Summit, NJ, USA, 1993.

[17] S. Colebourne, B. S. O'Neill, and others. Joda-Time. SourceForge Source Code Management, 2011. `http://joda-time.sourceforge.net/` (cited Jun 26, 2013).

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, second edition, 2001.

[19] T. T. Elvins. VisFiles: Presentation techniques for time-series data. *SIGGRAPH Comput. Graph.*, 31(2):14–16, 1997.

[20] J.-D. Fekete. The InfoVis toolkit. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pages 167–174, 2004.

[21] J.-D. Fekete, P.-L. Hemery, T. Baudel, and J. Wood. Obvious: A meta-toolkit to encapsulate information visualization toolkits–One toolkit to bind them all. In *Proc. 2011 IEEE Conf. Visual Analytics Science and Technology (VAST)*, pages 91–100, 2011.

[22] A. G. Forbes, T. Höllerer, and G. Legrady. behaviorism: A framework for dynamic data visualization. *IEEE Trans. Visualization and Computer Graphics*, 16(6):1164–1171, 2010.

[23] A. U. Frank. Different types of "Times" in GIS. In M. J. Egenhofer and R. G. Golledge, editors, *Spatial and Temporal Reasoning in Geographic Information Systems*, pages 40–62. Oxford University Press, New York, 1998.

[24] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman, Boston, MA, USA, 1995.

[25] D. Gilbert, T. Morgner, and others. JFreeChart. SourceForge Source Code Management, 2000. `http://sourceforge.net/projects/jfreechart/` (cited Jun 26, 2013).

[26] I. A. Goralwalla, Y. Leontiev, M. Özsu, D. Szafron, and C. Combi. Temporal granularity: Completing the puzzle. *Journal of Intelligent Information Systems*, 16:41–63, 2001.

[27] I. A. Goralwalla, M. T. Özsu, and D. Szafron. An object-oriented framework for temporal data models. In O. Etzion, S. Jajodia, and S. Sripada, editors, *Temporal Databases: Research and Practice*, LNCS 1399, pages 1–35. Springer, Berlin, 1998.

[28] R. L. Harris. *Information Graphics: A Comprehensive Illustrated Reference*. Oxford University Press, New York, 1999.

[29] S. Havre, E. Hetzler, P. Whitney, and L. Nowell. ThemeRiver: Visualizing thematic changes in large document collections. *IEEE Trans. Visualization and Computer Graphics*, 8(1):9–20, 2002.

[30] J. Heer and M. Agrawala. Software design patterns for information visualization. *IEEE Trans. Visualization and Computer Graphics*, 12(5):853–860, 2006.

[31] J. Heer and M. Bostock. Declarative language design for interactive visualization. *IEEE Trans. Visualization and Computer Graphics*, 16(6):1149–1156, 2010.

[32] J. Heer, S. K. Card, and J. A. Landay. prefuse: A toolkit for interactive information visualization. In *Proc. SIGCHI Conf. Human Factors in Computing Systems (CHI)*, pages 421–430. ACM, 2005.

[33] D. F. Huynh, D. R. Karger, and R. C. Miller. Exhibit: Lightweight structured data publishing. In *Proc. 16th Int. Conf. World Wide Web (WWW)*, pages 737–746. ACM, 2007.

[34] C. Jensen et al. The consensus glossary of temporal database concepts – February 1998 version. In O. Etzion, S. Jajodia, and S. Sripada, editors, *Temporal Databases: Research and Practice*, pages 367–405. Springer, Berlin, 1998.

[35] D. Keim, J. Kohlhammer, G. Ellis, and F. Mansmann, editors. *Mastering The Information Age – Solving Problems with Visual Analytics*. Eurographics, Goslar, Germany, 2010.

[36] R. Kosara and S. Miksch. Metaphors of movement: A visualization and user interface for time-oriented, skeletal plans. *Artificial Intelligence in Medicine*, 22(2):111–131, 2001.

[37] M. A. Kuhail and S. Lauesen. Customizable visualizations with formula-linked building blocks. In *Proc. Int. Conf. Computer Graphics Theory and Applications & Int. Conf. Information Visualization Theory and Applications (GRAPP/IVAPP)*, pages 768–771, 2012.

[38] M. A. Kuhail, K. Pandazo, and S. Lauesen. Customizable time-oriented visualizations. In G. Bebis, R. Boyle, B. Parvin, D. Koracin, C. Fowlkes, S. Wang, M.-H. Choi, S. Mantler, J. Schulze, D. Acevedo, K. Mueller, and M. Papka, editors, *Advances in Visual Computing, Proc. ISVC 2012, Part II*, LNCS 7432, pages 668–677. Springer, 2012.

[39] T. Lammarsch, W. Aigner, A. Bertone, J. Gärtner, E. Mayr, S. Miksch, and M. Smuc. Hierarchical temporal patterns and interactive aggregated views for pixel-based visualizations. In *Proc. 13th Int. Conf. Information Visualisation (IV)*, pages 44–50. IEEE, 2009.

[40] T. Lammarsch, W. Aigner, A. Bertone, S. Miksch, and A. Rind. Towards a concept how the structure of time can support the visual analytics process. In S. Miksch and G. Santucci, editors, *Proc. Int. Workshop Visual Analytics (EuroVA) in conjunction with EuroVis*, pages 9–12. Eurographics, 2011.

[41] T. Lammarsch, W. Aigner, A. Bertone, S. Miksch, and A. Rind. Mind the time: Unleashing the temporal aspects in pattern discovery. In M. Pohl and H. Schuman, editors, *Proc. 4th Int. Eurovis Workshop on Visual Analytics held in Europe (EuroVA)*, 2013.

[42] R. D. Peng and L. J. Welty. The NMMAPSdata package. *R News*, 4(2):10–14, 2004.

[43] C. Plaisant, R. Mushlin, A. Snyder, J. Li, D. Heller, and B. Shneiderman. LifeLines: Using visualization to enhance navigation and analysis of patient records. In *Proc. AMIA Symp.*, pages 76–80, 1998.

[44] H. Reijner. The development of the horizon graph. In L. Bartram, M. Stone, and D. Gromala, editors, *Proc. Vis08 Workshop From Theory to Practice: Design, Vision and Visualization*, 2008.

[45] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proc. IEEE Symp. Visual Languages (VL)*, pages 336–343, 1996.

[46] S. F. Silva and T. Catarci. Visualization of linear time-oriented data: a survey (extended version). *Journal of Applied System Studies – Special Issue on Web Information Systems Applications*, 3(2), 2002.

[47] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Trans. Visualization and Computer Graphics*, 8(1):52–65, 2002.

[48] J. J. Thomas and K. A. Cook, editors. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE, 2005.

[49] B. Urgun, C. E. Dyreson, R. T. Snodgrass, J. K. Miller, N. Kline, M. D. Soo, and C. S. Jensen. Integrating multiple calendars using tauZaman. *Software: Practice and Experience*, 37(3):267–308, 2007.

[50] M. Wattenberg. Arc diagrams: Visualizing structure in strings. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pages 110–116. IEEE, 2002.

[51] C. Weaver. Building highly-coordinated visualizations in Improvise. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pages 159–166, 2004.

[52] C. Weaver, D. Fyfe, A. Robinson, D. Holdsworth, D. Peuquet, and A. M. MacEachren. Visual analysis of historic hotel visitation patterns. In *2006 IEEE Symp. Visual Analytics Science And Technology (VAST)*, pages 35–42, 2006.