FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

A Software Framework for Visual Analytics of Time-Oriented Data

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Information & Knowledge Management

eingereicht von

Alexander Rind
Matrikelnummer 9825992

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.-Prof. Mag. Dr. Silvia Miksch
Mitwirkung: Dipl.-Inf. Dr. Tim Lammarsch

Wien, 21.11.2017

(Alexander Rind) (Silvia Miksch)

Erklärung zur Verfassung der Arbeit

Alexander Rind
Kleegasse 9
3483 Wagram am Wagram

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit einschließlich Tabellen, Karten und Abbildungen, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 21.11.2017

(Alexander Rind)

Acknowledgments

I want to thank my colleagues Tim Lammarsch, Wolfgang Aigner, Bilal Alsallakh, and Silvia Miksch for the inspiring discussions, the many hours of coding, and the constructive feedback that ultimately made TimeBench possible. In particular, I want to express my gratitude to my advisors Silvia Miksch and Tim Lammarsch for always finding the time and patience to give feedback despite their tight schedules.

I learned a lot about time and its complexities from Silvia Miksch, Wolfgang Aigner, and Tim Lammarsch and I also learned a lot from my colleagues Alessio Bertone, Paolo Federico, Theresia Gschwandtner, Bilal Alsallakh, Markus Bögl, Albert Amor, Christian Bors, Markus Wagner, Christina Niederer, Kerstin Blumenstein, and Florian Grassinger. I want to thank my friends Jörg and Andreas for encouraging me to start this Master's degree, and my friend Rudi for pursuing it to completion.

Finally, I dedicate this thesis to my family. Without their help and patience, I would not have had the energy to work on this thesis. For example, Theodor had to remind me repeatedly that I should not be working when the sun is not shining.

Abstract

Visual Analytics is one effective way to improve the understanding of large and complex datasets through the systematic combination of interactive visualizations and automated analysis techniques. Time-oriented data are highly relevant to many application fields of Visual Analytics but time and time-oriented data do also have a complex semantic structure involving design aspects such as granularities of time, different time primitives, and indeterminacy. Still, most existing software frameworks for visualization and Visual Analytics support only a subset of these design aspects to satisfy concrete application demands. However, for prototyping in basic research on Visual Analytics methods, a software framework is needed that supports the design aspects of time-oriented data in a systematic, theory-driven way.

Tackling such need, this work investigates how a software framework can support Visual Analytics of time-oriented data in an expressive and efficient way. Its outcomes comprise substantial parts of the conceptual software architecture and the prototypical implementation of TimeBench. In particular, this work focuses on TimeBench's data structures and import/export functions.

Development was guided by three desiderata on expressiveness regarding design aspects of time-oriented data and three desiderata regarding efficiency for software developers. The software architecture and implementation were designed based on established software design patterns for visualization such as the Proxy Tuple pattern. The resulting data structures are conceptually based on two interlinked direct acyclic graphs and implemented on top of the relational data tables provided by the prefuse framework. For evaluation regarding its desiderata, the work presents the reimplementation of two complex visualization techniques and two development case studies. These results demonstrate that TimeBench is useful for Visual Analytics prototyping with a focus on time-oriented data well.

Kurzfassung

Visual Analytics ist eine effektive Herangehensweise um große beziehungsweise komplexe Datenbestände zu verstehen. Dazu werden interaktive Visualisierungen von Daten und automatische Datenanalysemethoden planvoll verbunden. Zeitbezogene Daten sind höchst relevant in vielen Anwendungsgebieten von Visual Analytics, aber Zeit und zeitbezogene Daten haben auch eine komplexe semantische Struktur, die Designaspekte wie Granularitäten, verschiedene Zeitprimitive und Unbestimmtheiten einschließt. Doch die meisten bestehenden Software Framework für Visualisierung und Visual Analytics unterstützen nur Teile dieser Designaspekte, um konkreten Bedarf aus einzelnen Anwendungsgebieten zu decken. Gerade aber für das Prototyping in der Visual Analytics Grundlagenforschung ist ein Software Framework nötig, das die Designaspekte von zeitbezogenen Daten auf eine systematische, theoriebasiert Weise unterstützt.

Um diesen Bedarf aufzugreifen, untersucht diese Arbeit, wie ein Software Framework Visual Analytics mit zeitbezogenen Daten auf ausdrucksvolle und effiziente Weise unterstützen kann. Die Ergebnisse dieser Untersuchung stellen grundlegende Teile der konzeptuellen Softwarearchitektur und der prototypischen Implementierung von TimeBench dar. Vor allen stehen bei dieser Arbeit die Datenstrukturen und Import/ Export Funktionen von TimeBench im Mittelpunkt.

Die Entwicklung orientierte sich an drei Desiderata zur Expressivität hinsichtlich von Designaspekten zeitbezogener Daten und drei Desiderata zur Effizienz für Softwareentwicklerinnen und -entwickler. Die Softwarearchitektur und die Implementierung basieren auf etablierten Software Entwurfsmustern für Visualisierung wie dem Proxy Tuple Muster. Die resultierende Datenstruktur besteht konzeptuell aus zwei verknüpften gerichteten zyklenfreien Graphen und wurde mit Hilfe von relationalen Datentabellen aus dem prefuse Framework umgesetzt. Zur Evaluierung hinsichtlich der Desiderata erläutert die Arbeit wie zwei komplexe Visualisierungstechniken mit TimeBench umgesetzt werden können. Außerdem werden zwei Fallstudien über die Anwendung in Entwicklungsprojekten beschrieben. Diese Ergebnisse legen dar, dass TimeBench für das Visual Analytics Prototyping mit einem Fokus auf zeitbezogene Daten nützlich ist.

Contents

1 Introduction

Visualization is one effective way to improve the understanding of large and complex datasets [Card et al., 1999] and, thus, computer-based visualization systems are important tools to cope with data deluge in our generation's work and personal life. Interaction is an inherent part of visualization allowing users to apply their expert knowledge and tackle ill-defined problems. The growing research field *Visual Analytics* combines such interactive visualizations with automated analysis techniques and puts special focus on cognitive aspects of understanding, reasoning, and decision making on the basis of very large and complex datasets [Keim et al., 2010; Thomas and Cook, 2005].

Time is common and plays an important role in many application fields for Visual Analytics (e.g., in business [Lammarsch et al., 2009] or healthcare [Aigner et al., 2012; Rind et al., 2017]). Following Aigner et al. [2011b, p. 2], "data that are inherently linked to time" will be referred to as *time-oriented data*. Time is not just another 'flat' quantitative data dimension but has a complex semantic structure that is comprised of a hierarchy of granularities (e.g., years, months, days). Granularities can be combined to form different calendar systems (e.g., Gregorian, financial, academic) and capture natural cycles or social re-occurrences. Furthermore, data can relate to time using different time primitives: instants, intervals, and spans. Thus, it is worthwhile to treat time-oriented data specifically in order to provide appropriate Visual Analytics methods [Aigner et al., 2011b].

However, most existing *software frameworks* for visualization and Visual Analytics address time in a simplistic manner, usually in the form of equally spaced time series

(e.g., one value by month) or time stamped values (e.g., using Unix epoch). Visualization techniques and Visual Analytics solutions that demand advanced aspects of time-oriented data (e.g., GROOVE [Lammarsch et al., 2009], VisuExplore [Rind et al., 2011]) need to implement these aspects in an ad-hoc fashion. Therefore, the Visual Analytics community would profit from a reusable software framework for time-oriented data.

Research Questions

Thus, the research objectives of this thesis will be to investigate software design patterns for Visual Analytics of time-oriented data and to combine them in an implemented software framework. The main research question of this thesis is:

> *How can a software framework support Visual Analytics of time-oriented data in an expressive and efficient way?*

The sub-questions tackle two particular packages within such a framework:

> *Which data structure can manage time-oriented data that is linked to heterogeneous, hierarchically composed time primitives?*
>
> *How can text files with comma-separated values be annotated for configurable import of time-oriented data?*

1.1. Expected Results

The outcomes of this work will be a substantial part of the *software architecture and the prototypical implementation of **TimeBench**, a framework for Visual Analytics of time-oriented data*. The framework will encompass data structures, import/export functions, visual representations, interaction techniques, and automated analysis operations. It will be built based on the Information Visualization Reference Model [Card et al., 1999] and other software design patterns for information visualization [Heer and Agrawala, 2006]. For example, the Data Column pattern will be used for efficient storage in tables while the Proxy Tuple pattern will provide a convenient object-oriented programming interface. The data structure for time-oriented data will be based on conceptual proposals by Aigner [2006] and Lammarsch et al. [2011].

TimeBench will be primarily used for rapid prototyping in Visual Analytics research projects. Expressive time and data abstractions plus an extensible set of Visual Analytics operations should provide a platform to quickly recombine approaches and test ideas. Furthermore, the framework will be used in the FWF-sponsored project HypoVis for modeling hypotheses supported by the structure of time [Lammarsch et al., 2011].

1.2. Methodological Approach

Following the nested model for visualization design and validation by Munzner [2009] (Figure 1.1), the contributions of this work lie mainly in the innermost layer "algorithm design" and partly in "encoding/interaction technique design".

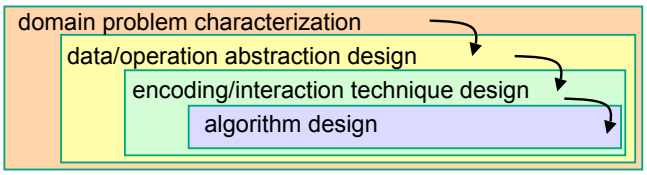


Figure 1.1.: Nested model for visualization design and validation [Munzner, 2009]

On the *algorithm* layer software design and implementation are needed for the data structures and the operations powering visual representation, interaction, and automated analysis. The design will follow established software design patterns for visualization (e.g., [Heer and Agrawala, 2006]) and time-oriented data (e.g., interval tree [Cormen et al., 2001]). The implementation will target the Java programming language and will be built on top of the open-source visualization framework prefuse [Heer et al., 2005]. To mitigate the threat of incorrect algorithms, the library will be thoroughly tested.

On the *encoding/interaction technique* layer visual representation and interaction techniques for time-oriented data will be developed. The design of these techniques will start from existing work but their implementation will build on top of the framework's algorithmic layer. Qualitative result image analysis serves as further downstream validation for correctness.

The outer layers are tackled insofar that results from previous research inform on the needs of domain users and several simple Visual Analytics prototypes (e.g., horizon graph with indexing [Reijner, 2008], PlanningLines [Aigner et al., 2005],) will be built as application examples to demonstrate the capabilities of the framework. Two case studies will report about the application of TimeBench in two student projects Further validation will result from its adoption in HypoVis and other research projects.

1.3. Collaboration in Research Projects

TimeBench was designed and developed collaboratively by a team of researchers. Besides Alexander Rind, the author of this thesis, Tim Lammarsch, Wolfgang Aigner, Bilal

Alsallakh, and Silvia Miksch were in the TimeBench core team [Rind et al., 2013]. The author of this thesis was involved in all design decisions and was the lead developer for the data structures and import/export functions of TimeBench. He also implemented a number of Visual Analytics operations and TimeBench demos.

While this thesis needs to provide a complete overview of the TimeBench framework, it presents the author's contributions in more detail (esp. Sections 4.3 and 4.4). The contributions of others are clearly attributed throughout the thesis.

The core team published a journal article on TimeBench [Rind et al., 2013] with the student as first author. He also presented the article at the IEEE VAST conference.

1.4. Structure of the Work

Chapter 2 surveys software frameworks for Visual Analytics of time-oriented data and other related work. Due to the non-existence of a framework addressing the time abstraction aspects collected by Aigner et al. [2011b], Chapter 3 establishes three desiderata on expressiveness. Additionally, it postulates three desiderata on efficiency and lists some of the essential software features.

The design rationales and software architecture of TimeBench are presented in Chapter 4. First, the fundamental decisions for a polylithic framework [Bederson et al., 2004; Fekete, 2013] based on design patterns [Heer and Agrawala, 2006] and the choice of prefuse [Heer et al., 2005] base framework are justified. After a summary of the calendar package, Section 4.3 elaborates the data structures on the conceptual and the implementation level. Further sections introduce the import and export functions, automated analysis operations, and visual representation and interaction techniques.

Chapter 5 presents two demonstration applications showcasing reimplementations of horizon graph [Reijner, 2008] and a PlanningLines [Aigner et al., 2005]. Furthermore, two case studies of students applying respectively extending TimeBench are reported. The fulfillment of desiderata is discussed in Chapter 6. Finally, Chapter 7 concludes the work by answering the research questions and pointing toward directions for future research and development.

2
Related Work

Visualization of time-oriented data is a very actively researched topic. More than 100 visualization technique are collected in the book "Visualization of Time-Oriented Data" [Aigner et al., 2011b]. Time also plays an important role in Visual Analytics, which is demonstrated by the dedicated chapter in "Mastering the Information Age" [Keim et al., 2010].

2.1. Visualization Frameworks

Visual Analytics developers looking for a prototyping software environment have a wide range of options (Figure 2.1). First, they can work within a data analysis and visualization tool like Tableau/Polaris [Stolte et al., 2002], or Keshif [Yalçin et al., 2016]. Second, they can develop a prototype based on a software framework such as prefuse/Flare [Heer et al., 2005], IVTK [Fekete, 2004], behaviorism [Forbes et al., 2010], Tulip [Auber, 2004; Auber et al., 2012], ProtoVis [Bostock and Heer, 2009; Heer and Bostock, 2010], or D3.js [Bostock et al., 2011]. Some tools such as Improvise [Weaver, 2004] can also be used as a framework. The recent frameworks Vega [Satyanarayan and Heer, 2014; Satyanarayan et al., 2016] and Vega-Lite [Wongsuphasawat et al., 2016; Satyanarayan et al., 2017] are designed as a low-level grammar and a high-level grammar for declarative specification of interactive visualizations. Vega can be edited through the tool Lyra [Satyanarayan and Heer, 2014]. Vega-Lite is the foundation of Voyager [Wongsuphasawat et al., 2016], a visualization tool providing recommendations. Working

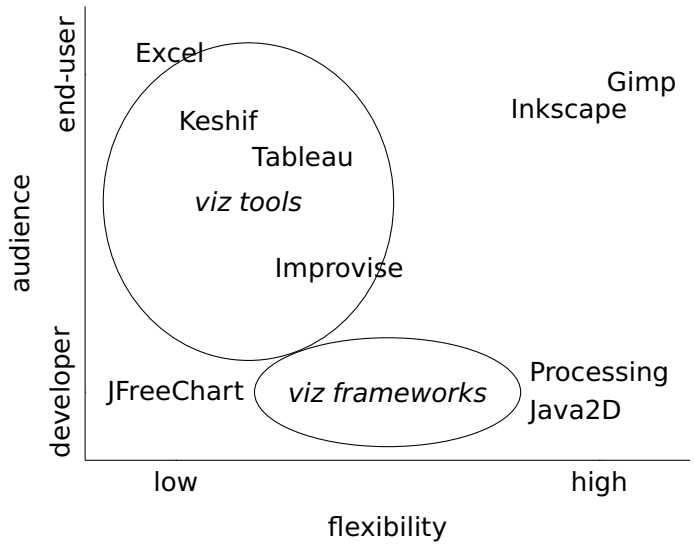


Figure 2.1.: Design space for visualization software environments.

with a tool is easier for end-users and typically requires less time for training and first prototypes, but there are often limitations of what results are possible and what can be automatized. Furthermore, these tools and frameworks are conceptualized with different levels of flexibility: On the one hand, graphics frameworks like Java2D or Processing [Reas and Fry., 2014] and graphics tools like Gimp or Inkscape allow complete flexibility but do not offer significant structure for turning data into visual representations. On the other hand, software environments that are too rigidly structured like Microsoft Excel or JFreeChart [Gilbert et al., 2000] will soon impose limitations to their users. Visualization frameworks aim for a balanced level of flexibility by providing extendable software features based on information visualization concepts such as visual channels [Card et al., 1999; Munzner, 2014, Chapter 5].

2.2. Time-Oriented Data in Visualization Frameworks

prefuse [Heer et al., 2005] is an open-source Java framework for information visualization. Its polylithic architecture is based on visualization design patterns and is composed of data tables, mapping of abstract data to visual structures, and the rendering to the display. This makes prefuse suitable for extensions. prefuse has a dedicated variable type for time, but it only stores the timestamp as a number with support for string parsing and formatting. Apart from that, support for time-oriented data needs to be implemented by application developers.

D3.js [Bostock et al., 2011] is a very widely used open-source framework implemented in JavaScript. Its core functionality focuses on generically mapping arbitrary data to document objects with a webpage. Yet, D3.js is accompanied by a large ecosystem of supplementary features. For example `d3.scaleTime()`[1] provides a linear time axis with axis labels based on calendar units. The d3-time package [2] supports calculations based on calendar units. Cubism.js[3] is an extension to D3.js showing real-time time series in a horizon graph [Reijner, 2008]. STRAD-Wheel [Fernández-Prieto et al., 2017] is an extension to D3.js that displays time series in a circular layout with facets for aggregation and filtering by different granularities of time. D3.js, however, does not provide a data structure for the development of visualizations of complex time-oriented data such as data with heterogeneous primitives.

Many frameworks support instant data on a linear time axis, e.g., IVTK [Fekete, 2004] or ProtoVis [Bostock and Heer, 2009]. Vega [Satyanarayan and Heer, 2014; Satyanarayan et al., 2016] and Vega-Lite [Wongsuphasawat et al., 2016; Satyanarayan et al., 2017] support a linear time axis, instants and intervals, and aggregation by some calendar units. Simile TimeLine [Huynh et al., 2007] is specialized component for visualizing interval data. Improvise provides the "reruns" component [Weaver et al., 2006] that displays an instant's data a glyph in a matrix. By changing the number of glyphs in a matrix row, the users can explore cyclical patterns in time.

Kuhail and Lauesen [2012] present a visualization tool with time-oriented building blocks [Kuhail et al., 2012] that can be glued together and customized by spreadsheet-like formulas. This approach allows designers to create several advanced visualizations for time-oriented data such as horizon graphs, cycle plots, or spiral graphs without the need to learn a programming language. Even though the blocks are largely customizable, the expressiveness is a limited by the available blocks and properties (e.g., there is a dedicated building block for a cycle plot).

Nanocubes [Lins et al., 2013] are data structures for efficient exploration of aggregated data from big spatio-temporal datasets. They allow binning by some calendar units.

2.3. Frameworks for Time and Calendars

Besides visualization and visual analytics, there are software frameworks specialized on time and calendars: The standard Java base classes include an implementation of

[1] https://github.com/d3/d3-scale/blob/master/README.md#scaleTime, last accessed Nov 2, 2017
[2] https://github.com/d3/d3-time, last accessed Nov 2, 2017
[3] https://square.github.io/cubism/, last accessed Nov 2, 2017

the Gregorian calendar.[4] Joda Time [Colebourne et al., 2011] provides calendar systems from different parts of the world. tauZaman [Urgun et al., 2007] is supports multiple calendar systems with multiple calendars, and multiple granularities. It is built as a distributed system, where clients send requests such as conversion between two calendars to a tauZaman server. Additional calendar systems can be defined by XML or custom Java code depending on calendar system's complexity. Moment.js Luxon [5] is a JavaScript framework that can parse, validate, manipulate, and display dates and times. It supports instant, interval, and span primitives.

2.4. Summary

Most if not all visualization frameworks support time in one way or another. Applying calendar units to explore cyclical patterns in time is also possible in some of the frameworks but typically there are only a few calendar units of the Gregorian calendar supported. Most frameworks support instants and some support intervals, but typically, these time primitives are not provided together. Neither could other time primitives such as an indeterminate interval be found.

Overall, existing software frameworks for visualization and Visual Analytics address time and time-oriented data on an ad-hoc basis. Their support satisfies typical needs from scientific and commercial Visual Analytics applications but they do not tackle the design aspects of time-oriented data in a systematic, theory-driven way. However, basic research on Visual Analytic methods for time-oriented data needs such a broad systematic software framework to rapidly prototype and test novel methods.

[4] https://docs.oracle.com/javase/7/docs/api/java/util/GregorianCalendar.html, last accessed Nov 2, 2017

[5] http://moment.github.io/luxon/, last accessed Nov 20, 2017

3
Framework Requirements

As the analysis of related work (Section 2.1) demonstrated, existing software frameworks for visual analytics support time-oriented data to some extent but do so in a rather simplistic manner without addressing the multiple aspects of time-oriented data [Aigner et al., 2011b] needed for advanced visual analytics solutions.

As a roadmap for the development of TimeBench, we established a list of desiderata that TimeBench needs to address and structured these along the lines of expressiveness (Section 3.1) and efficiency (Section 3.2). Additionally we collected specific software features (Section 3.3) that software developers would frequently need when working with time-oriented data. By addressing these needs, TimeBench can be of larger practical utility and decrease its adoption costs.

Method We collected and ranked the requirements for TimeBench in a series of focus group meetings. The participants of the focus group meetings were members of the visual analytics research projects HypoVis (FWF-funded, #P22883) and CVAST (a Laura Bassi Centre of Excellence, #822746). All participants had experience in visual analytics with time-oriented data and two participants were co-authors of a reference book on "Visualization of Time-Oriented Data" [Aigner et al., 2011b]. One participant had designed a framework for visualization of time-oriented data on the conceptual level as part of his doctoral dissertation [Aigner, 2006, Chapter 10]. This conceptual framework served as inspiration for TimeBench, which should translate these concepts to an implemented software framework and revise them where applicable.

3.1. Desiderata on Expressiveness

For TimeBench to be an expressive software framework for time-oriented data, it needs to support in particular the three time abstraction aspects described by Aigner et al. [2011b]: (1) granularity & calendars, (2) time primitives, and (3) determinacy.

Granularity & calendars Calendars are used to divide time into units that are understood by humans such as the months or hours. Granularities and granules generalize such time units and have been formally defined by Bettini et al. [2000]. This aspect of time abstraction can be applied to segment the time axis into human-readable units, to aggregate time-oriented data accordingly, and to construct advanced visualization techniques [Lammarsch, 2010] such as cycle plot [Cleveland, 1993] (Figure 3.1) or GROOVE [Lammarsch et al., 2009].

Time primitives Data can be associated to either a point in time, the range between two time points, or a given duration. Goralwalla et al. [1998] conceptualize these associations as time primitives and defines these three cases as instant (e.g., the day granule 31 July 2017), interval (e.g., the range of days between 4 and 14 July 2017), and span (e.g., a duration of 10 days). While instants and intervals have fixed positions in time and are called anchored primitives, spans are unanchored and provide no information about their position in time. In a visual analytics system, the difference between primitives need to be addressed accordingly. For example, while instants have only three possible order relations (before, identical, after), intervals can have 13 qualitative relations (e.g., starting together and ending earlier) [Allen, 1983]. The combination of heterogeneous time primitives

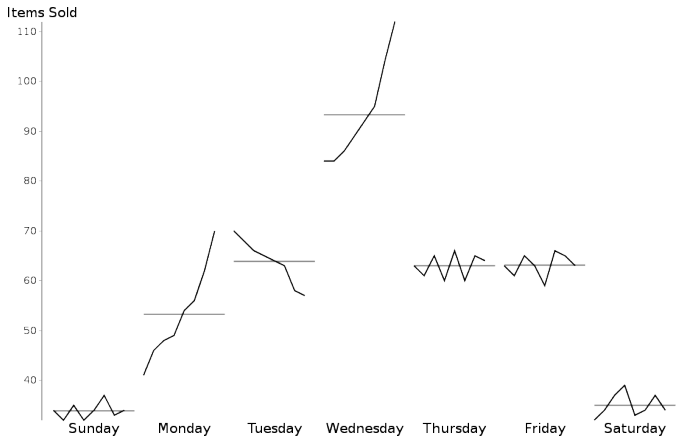


Figure 3.1.: Cycle plot example *Screenshot by the author, based on code by Graham Odds*

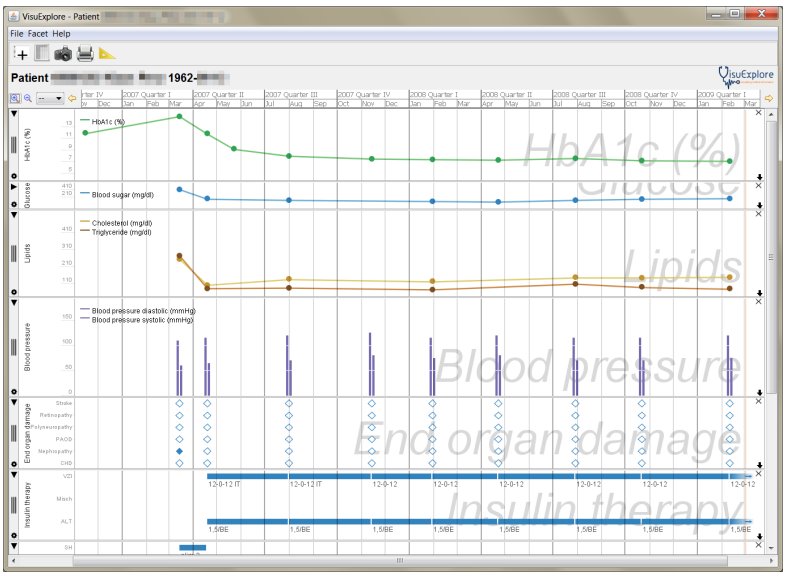


Figure 3.2.: VisuExplore prototype [Rind et al., 2011] with different visual representation methods for data associated to instants and intervals

Reused under CC-BY-ND license from http://ieg.ifs.tuwien.ac.at/projects/VisuExplore/

is of particular relevance in healthcare [Aigner et al., 2012; Rind et al., 2017] as demonstrated in the VisuExplore design study [Rind et al., 2011] (Figure 3.2).

Determinacy Applications in healthcare also illustrate the need to consider temporal indeterminacy [Aigner et al., 2012; Kosara and Miksch, 2001]. For example, a clinical guideline could prescribe that treatment has to be within one to three weeks before surgery and last between two and four days but leave exact start point and duration undecided, so that caregivers can adjust it based on patient status. Such indeterminate intervals were demonstrated in the PlanningLines design study [Aigner et al., 2005] (Figure 3.3). Another source of indeterminacy can be the conversion between granularities (e.g., when monthly data are combined with weekly data).

These three time abstraction aspects are the top priority for the expressiveness of TimeBench. State-of-the-art visual analytics frameworks already address a number of design aspects of time-oriented data such as quantitative/qualitative scale, abstract/spatial frame of reference, different number of variables. Other design aspects such as multiple perspectives or streaming data would exceed the scope of this work and need to be addressed in the future.

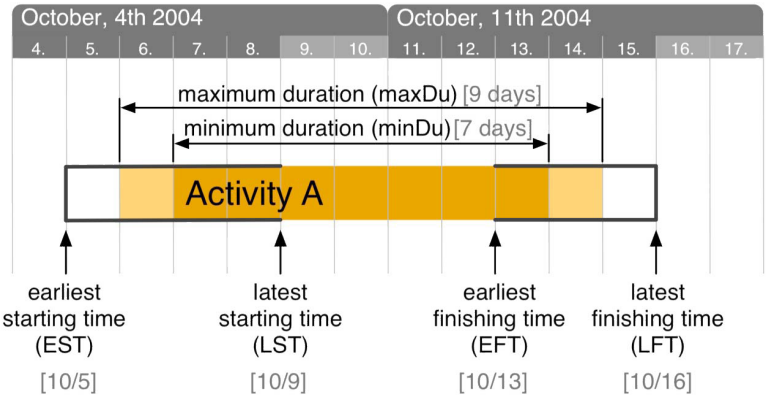


Figure 3.3.: Temporal constraints of the indeterminate interval as represented in the PlanningLines glyph [Aigner et al., 2005] © 2005 IEEE

3.2. Desiderata on Efficiency

Besides being expressive for design aspects of time-oriented data, TimeBench needs to be an efficient basis for software development. Therefore it should be (1) built around a common data structure, (2) accessible for developers, and (3) efficient at runtime.

Common data structure TimeBench should not only support different forms of time-oriented data, it also needs to support the combined analysis of heterogeneous data. In particular, the researchers engaged in the HypoVis project planned to apply TimeBench as the technical foundation to model hypotheses supported by the structure of time within a Visual Analytics environment as conceptually described by Lammarsch et al. [2011]. Consequently, standardized data structures should be at its core.

Developer accessibility Since the "users" of a software framework are software developers, it important that they can learn the framework quickly and can use it without barriers [Heer et al., 2005]. Thus, the framework's application programming interface should be designed to match metaphors of the time domain. Furthermore, the framework should be modular so that can be extended further. A polylithic architecture [Bederson et al., 2004; Fekete, 2013] should allow for the composition of visual analytics operations both from the TimeBench base components and future extensions.

Runtime efficiency Visual analytics solutions built with TimeBench need to provide interactive reaction times (cp. [Munzner, 2014, p. 139 ff.]). Therefore, its data struc-

tures and operations should be optimized based on the aspects of time-oriented data.

3.3. Essential Software Features

TimeBench is to be designed as an extensible and polylithic software framework. In order to provide a good basis for software development, a number of software features need to be available within the base framework instead of being part of extensions.

Data structures TimeBench should provide data structures for determinate and indeterminate time primitives and for multivariate data of quantitative and qualitative types associated with these primitives [Aigner et al., 2011b]. It should also be possible to structure these data items as a network with edges or as a tree with hierarchy [Munzner, 2014].

Import/export functions Time-oriented data needs to be imported from structured text files such as comma-separated values (CSV) and from calendars in the iCalendar format. Furthermore, a file format is needed that can store all details of the TimeBench data structure so that data can be exported and imported without information-loss.

Visual representations A linear time axis plus generic visual representations such as marks, lines, and bars need to be available to create, for example, scatter plots, line plots, and Gantt charts.

Interaction techniques Zoom and pan needs to be available for the linear time axis. In addition, generic interaction techniques like detail-on-demand and brushing and linking [Shneiderman, 1996] should be available.

Automated analysis operations Filtering and aggregation based on time granularities should be possible. For example, show only "Fridays" or average all hourly values of each day. An indexing operation should transform all quantitative values into the changes relative to the indexing point [Aigner et al., 2011a], which is often applied in horizon graphs [Reijner, 2008].

3.4. Discussion of Requirements

Some of the requirements postulated above are in conflict with each other. For example, higher expressiveness implies that the framework will most likely be harder to learn for

developers and could be less efficient at runtime. While a common data structure is of benefit for developers, it could impair the runtime efficiency.

When finding suitable trade-offs, the desiderata should be weighted in the same order as presented in this work. Expressiveness is the primary desideratum because TimeBench is developed in particular for academic usage where rapid prototyping with a wide range of possible combinations is required.

4
Framework Design and Implementation

The previous chapter laid out the requirements for TimeBench. It should primarily be expressive to address the time abstraction aspects granularity & calendars, time primitives, and determinacy. Secondarily it should be an extensible framework based on top of a common data structure. It should also be easy to use for developers and efficient at runtime. The following sections, elaborate how TimeBench was designed and implemented in order to address these requirements.

4.1. Fundamental Design Decisions

In order to address the efficiency desiderata, TimeBench's software architecture is based on well-established design patterns for Information Visualization [Heer and Agrawala, 2006]. Thus, we expect it to be more easily extensible, more accessible to software developers, and to profit from experience regarding runtime efficiency. The general architecture of a visual analytics solution using TimeBench will follow the reference model for visualization [Card et al., 1999], which is a special form of the Model–View–Controller design pattern [Heer and Agrawala, 2006]. In this model, raw data is first transformed to abstract data, then enriched with visual attributes, and finally rendered onto views, whereby the user can adjust settings at each transition via interaction techniques.

A polylithic visualization framework [Bederson et al., 2004; Fekete, 2013] supporting the reference model provides various implemented operations at each transition. A solution designer can combine these operations in multiple ways similar to Lego bricks.

For example, a scatter plot would be a composition using a CSV reader, a horizontal and a vertical position mapping, and a renderer for marks. All that is needed to extending the scatter plot to a bubble chart would be to add a size mapping. Likewise, TimeBench can provide a collection of operations that are specialized on time-oriented data such as a reader for iCalendar files or a calendar-based position mapping (cp. Section 3.3).

The implementation of TimeBench is designed as an extension of prefuse [Heer et al., 2005], which is a general purpose software framework for Information Visualization written in Java. prefuse has a polylithic architecture and is based on design patterns in accordance to what is planned for TimeBench. The design decision to extend prefuse rather than building TimeBench from scratch was supported by the following arguments:

(1) In general, software developers would rather accept a comparatively smaller extension than a completely new framework, as they have to consider the time needed to learn the framework and to adapt their existing code base. In particular, the developers within the research projects HypoVis and CVAST are familiar with prefuse and, at the time when TimeBench development started, were predominately using this framework for their prototyping.

(2) Some previous visual analytics prototypes for time-oriented data such as CareVis [Aigner and Miksch, 2006] or VisuExplore [Rind et al., 2011] had been developed using prefuse. It was planned to reuse and generalize part of this code base like their calendar-based time axis labels.

(3) As prefuse has been available as free and open-source software on SourceForge since 2004[1] and has been widely adopted it can be assumed that a large number of bugs have been discovered and fixed. A newly written framework would need more time to reach the same level of maturity. Furthermore, prefuse contains several optimizations for runtime efficiency unrelated to time-oriented data such as graph layout.

(4) The meta-framework Obvious [Fekete et al., 2011] provide interoperability between various Visual Analytics software frameworks. Thus, prefuse/TimeBench components could be combined with the visualization frameworks IVTK [Fekete, 2004], Improvise [Weaver, 2004], or JUNG [The JUNG Development Team, 2006] and exchange data with databases via JDBC, Weka, or RapidMiner.

Roadmap prefuse provides a good selection of data structures and operations for multivariate data and for graph data but it lacks specific data structures and operations

[1] https://sourceforge.net/projects/prefuse/files/prefuse/, last accessed Oct 29, 2017

for time-oriented data. Thus, the primary core functionality of TimeBench needs to be a solid data structure for time-oriented data that addresses the expressiveness desiderata (cp. Section 3.1), which will be presented in Section 4.3. The second core functionality of TimeBench is calendar support, which will be summarized in Section 4.2. Furthermore, a good selection of operations is needed to provide software developers with frequently needed functionality. Yet, this selection need not be complete, because operations can easily be added due to TimeBench's polylithic architecture. These operations will be presented in Sections 4.4–4.5.

4.2. Calendars

While the author of this thesis focused on data structures (Section 4.3), the design and development of the calendar package was lead by Tim Lammarsch. Therefore details about calendar support in TimeBench are beyond the scope of this work; instead the journal article [Rind et al., 2013, p. 2251 f.] should be consulted. This section summarizes key concepts necessary to understand the data structures of TimeBench.

The calendar package is designed based on the calendar operations described by Bettini et al. [2000] and Goralwalla et al. [2001]. Time is modeled as a discrete domain composed of atomic units called chronons. In the context of a Java virtual machine, such chronons would be milliseconds. A mapping from chronons to subsets of chronons is called a granularity (e.g., days would be such a mapping) and a subset of chronons mapped by a granularity is called granule (e.g., 2 June 2017). Finally, granularities can have different mappings depending on which calendar they belong to. TimeBench encapsulates this calendar model in the classes `Granule`, `Granularity`, and `Calendar`.

A `Granularity` is defined by its identifier and its context identifier. For example a granularity with the identifier for day and the context identifier for week would map chronons to days of week (Monday, ..., Sunday, Monday, ...). In contrast a granularity with the identifier for day and the predefined context identifier `TOP` would map chronons to calendar day (3 May 2017, 4 May 2017, ...).

A `Granule` is defined by its identifier and the granularity it belongs to and it has a human readable label. For example in the day-of-week `Granularity`, the identifier 0 could stand for "Monday". Additionally, the `Granule` can have a context granule that further specifies its time. For example, a "Monday" `Granule` has the context granule "week 45", which has the context granule "2017". If the time of a `Granule` is fully specified it is a "particular granule", otherwise it is a "general granule". For a particular granule, it is possible to calculate the infimum and the supremum, which are its first and its last chronon. Vice versa, the `Granule` identifier and context granules can be calculated from the infimum or the supremum (or actually from any chronon within its

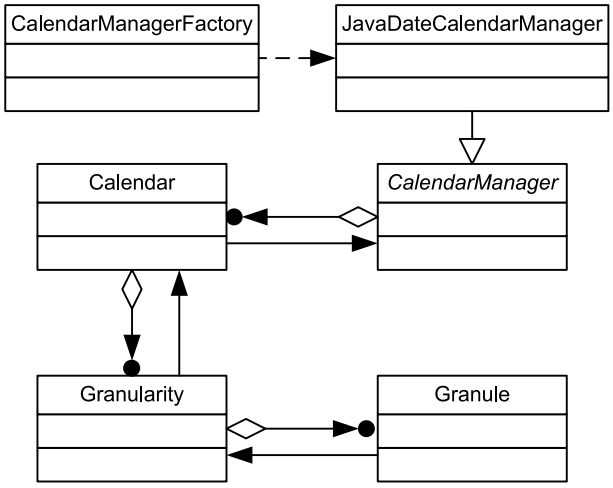


Figure 4.1.: Classes of the calendar package [Rind et al., 2013]
The diagram depicts classes and their relations using the extended object-modeling technique [Gamma et al., 1995; Heer and Agrawala, 2006]. The diagram is simplified and does not show all details.

duration). General Granules are useful to represent cyclic patterns or indeterminacies.

Most calendar operations can be realized by conversion between different Granularitys, conversion from chronons to Granules (in fact chronons are Granules of the predefined BOTTOM Granularity), and integer arithmetic with Granule identifiers.

The calendar classes are designed in a generic way as a Facade [Gamma et al., 1995] to a calendar backend that implements the CalendarManager interface (Figure 4.1). JavaDateCalendarManager is available as a base implementation using the GregorianCalendar included within the Java core packages, while future extensions based on τZaman [Urgun et al., 2007] or JodaTime [Colebourne et al., 2011] are possible.

4.3. Data Structures

TimeBench's data structures aim to achieve a desired expressiveness regarding time abstraction (cp. Section 3.1), but these data structures should also be accessible to developers in a meaningful way. Therefore, the data structures were first modeled in a conceptual way. Then an efficient backend data structure was built, which can be accessed and manipulated via an API that follows the conceptual model.

4.3.1. Conceptual Data Model

The data structures of TimeBench are inspired by conceptual ideas of Aigner [2006, Chapter 10]. In particular, he proposed a clear distinction between temporal and application-

specific data aspects of time-oriented data and he modeled TemporalObjects and TemporalElements as composite objects that could have TemporalObjects respectively TemporalElements as child objects (Figure 4.2).

TimeBench adopts these two architectural design decisions, because they make it possible to save heterogeneous time-oriented data in a common data structure. Each data item is modeled as a TemporalObject that stores application data together with a reference to a TemporalElement. More formally, this conceptual data structures can be modeled as a set of TemporalObjects, a set of TemporalElements an a timing function mapping the former to the later [Rind et al., 2013, p. 2249]. The abstract class TemporalElement provides a uniform interface for time primitives, which have very different internal structure:

- An Instant references a point in time, i.e., a Granule of a granularity [Lammarsch et al., 2011] (Figure 4.3a).
- A Span is defined a number of granules of a granularity [Lammarsch et al., 2011].
- An Interval could be given by its first and last Granule, by its first Granule and its length as number of granules, or by its last Granule and its length. Since such time primitives are already modeled as Instant and Span, TimeBench expresses

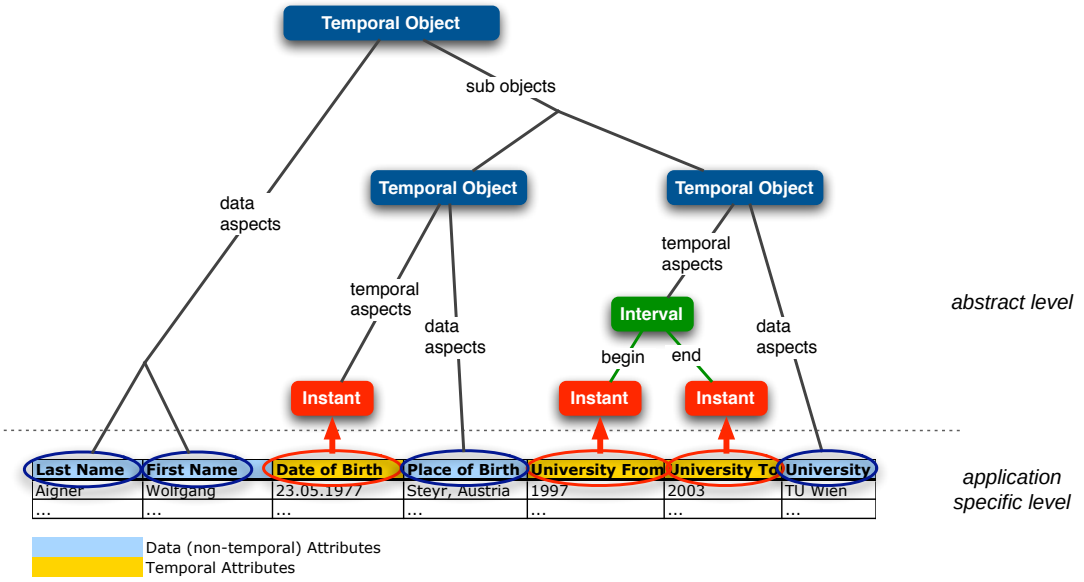


Figure 4.2.: Building temporal objects from a data table using the conceptual model by Aigner [2006, p. 218] *Image courtesy of Wolfgang Aigner*

the Interval as a composite primitive consisting of two Instants or an Instant and a Span (Figure 4.3b).

- Beyond these three time primitives, further time primitives are possible. For example, an indeterminate interval can be expressed as a composite primitive consisting of two Intervals for possible start and possible end and two Span for minimum and maximum duration (Figure 4.3c).
- Such composite time primitives can also be built as a set of child primitives in order to address particular requirements of an application scenario (Figure 4.3d). For example, a psychotherapeutic treatment series could be linked to a set of ten intervals.

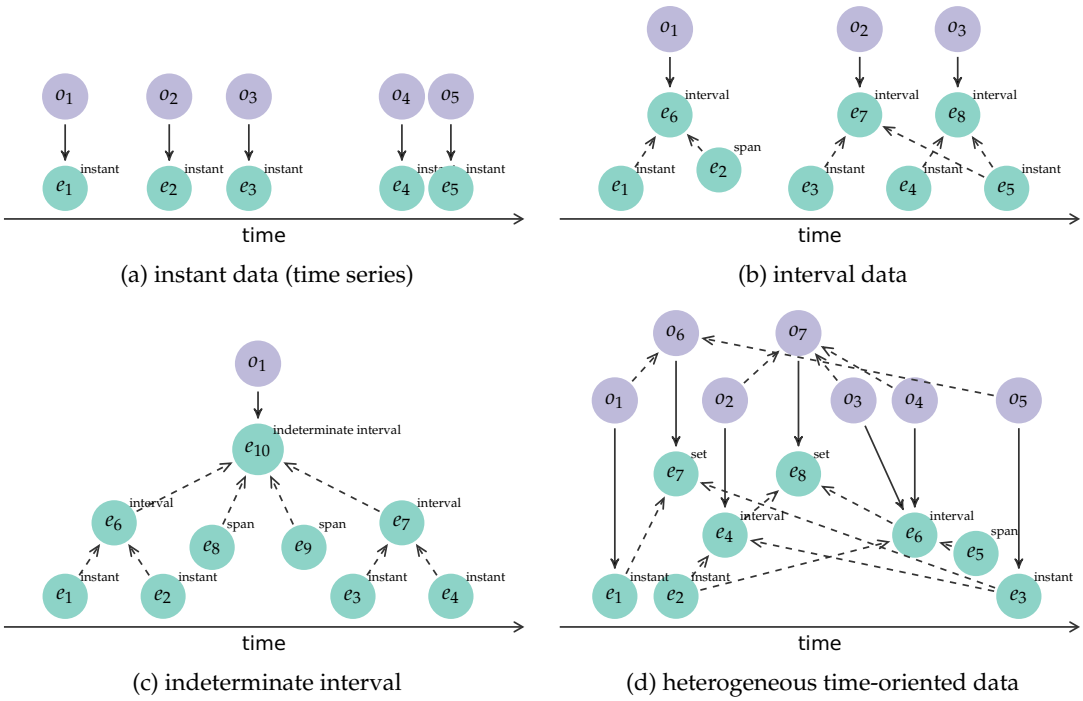
Thus, hierarchical composition of TemporalElements has the benefit that implementations specific to simple primitives can be reused within complex primitives. Composition of TemporalElements is modeled as a directed acyclic graph [Kerren et al., 2014, p. 2]:

- A parent element can have multiple child elements (e.g., in Figure 4.3b the Interval e_6 is composed of the Instant e_1 an the Span e_2).
- A child element can have multiple parent elements (e.g., in Figure 4.3b the Instant e_5 marks the end of the Intervals e_7 and e_8).

For consistency and to stay compatible with Aigner [2006]'s conceptual model, TemporalObjects can be composite objects organized in a directed acyclic graph structure as well.

4.3.2. Backend Data Structure

TimeBench stores TemporalObjects and TemporalElements in its data structure classes TemporalDataset and TemporalElementStore (Figure 4.4). Internally, they deposit temporal and domain-specific data in relational data tables that are provided by the prefuse framework. These Tables realize the Data Column design pattern [Heer and Agrawala, 2006], i.e., they manage their data not by rows but by columns. For example a discrete quantitative attribute would be stored as an integer array by the prefuse class IntColumn. Accessor methods of the Table use a row manager to map between row indices of the table and the column if these are different. The Data Column data structure has the advantages that data in a column are usually homogeneous and can, thus, be stored and indexed more efficiently. Furthermore. columns can be efficiently added or removed, and columns can be used by multiple tables (such as raw data and derived data) [Heer and Agrawala, 2006]. Since the TemporalDataset class extends a generic prefuse data



(a) instant data (time series)

(b) interval data

(c) indeterminate interval

(d) heterogeneous time-oriented data

Figure 4.3.: Four temporal dataset examples depicted at the conceptual level. Each green circle e_x represents temporal element, i.e., an object of a subclass of TemporalElement; each violet circle o_x represents a temporal object , i.e., an object of TemporalObject or a subclass. Dashed lines represent composition of complex such as an interval defined by a start instant and an end instant.

structure, its non-temporal aspects are fully compatible with existing prefuse operations. For example, a TemporalDataset of blood tests could be shown as a scatter plot of glucose by cholesterol without any TimeBench code.

While Tables are an efficient data structure and a convenient abstraction for designing the flow of data, accessing individual TemporalObjects or TemporalElements by their row index is cumbersome. To make these available in an object-oriented fashion, TimeBench applies the Proxy Tuple pattern [Heer and Agrawala, 2006], a variant of the Facade pattern [Gamma et al., 1995]. Following this design pattern, the so-called tuple does not store its data directly but acts as a proxy to a Table object at a stored row index. Listing 4.1 shows how this works in the prefuse class TableTuple. TemporalObjects and TemporalElements both extend class TableTuple with time-specific accessor methods as will be described below. As TemporalObjects and TemporalElements need a TemporalDataset and a TemporalElementStore to store their data, new objects are created from

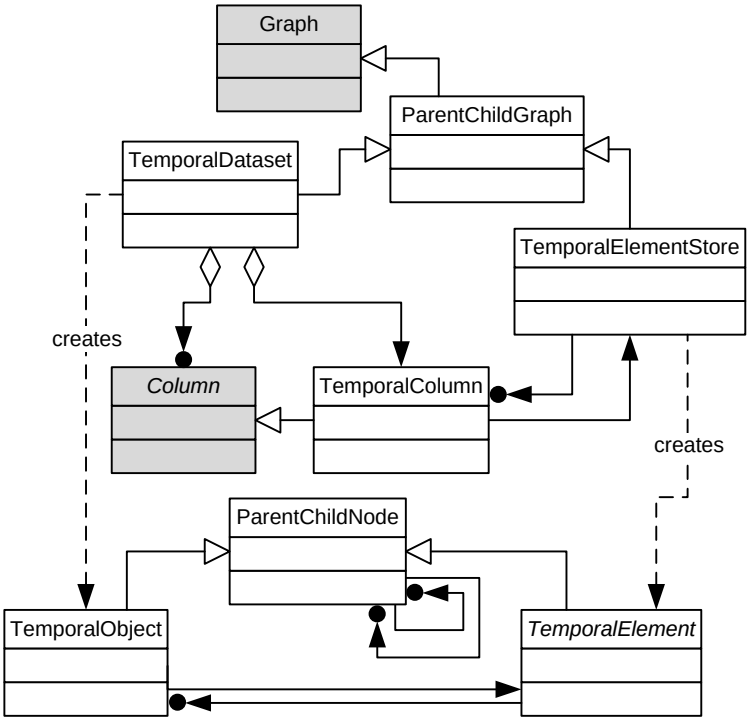


Figure 4.4.: Main data structures in TimeBench: Both TemporalDataset and TemporalElementStore are subclasses of ParentChildGraph, which is a subclass of the prefuse Graph [Heer et al., 2005]. They create TemporalObject and TemporalElement instances respectively. TemporalDataset and TemporalElementStore are linked by an instance of TemporalColumn, which is a subclass of a prefuse Column.
The diagram depicts classes and their relations using the extended object-modeling technique [Gamma et al., 1995; Heer and Agrawala, 2006]. The diagram is simplified and does not show all details.

factory methods provided by these data structures.

Since TemporalObjects and TemporalElements can be composed of a hierarchy of subobjects, their data structure needs to store the links between parent and child objects. For this, TimeBench extends the prefuse Graph class, which applies the Relational Graph design pattern [Heer and Agrawala, 2006]. According to this design principle, the Graph stores graph its nodes and edges in two Tables. For efficient lookup it provides indices. However, the generic accessor methods of Graph do not convey the semantics of composition in a directed acyclic graph. In order to improve developer accessibility and to avoid errors from mixing up edge direction, the classes ParentChildGraph and ParentChildNode extend Graph and TableNode with meaningful accessor methods (Listing 4.2). By convention, edges are directed from directed from child to

Listing 4.1: Excerpts of TableTuple from prefuse illustrating how accessor methods are realized as proxy to an underlying data structure.

```
public class TableTuple implements Tuple {
   protected Table m_table;
   protected int m_row = -1;

   public final Object get(String field) {
      validityCheck();
      return m_table.get(m_row, field);
   }
}
```

parent.

The connection from a TemporalDataset to its TemporalElementStore is built via a a special column type TemporalColumn. While all TemporalElements used in a TemporalDataset need to be from the same TemporalElementStore, a TemporalElementStore can be shared by multiple TemporalDatasets. This makes it possible to use the same TemporalElements for both raw and derived data.

4.3.3. Temporal Elements

TimeBench provides various subclasses of TemporalElement for different time primitives (Figure 4.5). On the first level of inheritance, AnchoredTemporalElements are distinguished from UnanchoredTemporalElement. These are further extended to Instant, Interval, and Span. Furthermore, each TemporalElement can be accessed as a GenericTemporalElement, which provides read-write access to the underlying data. Therefore, the methods asPrimitive() and asGeneric() allow the developer to switch between the GenericTemporalElement and the other subclasses. The corresponding objects are created and cached by the class TemporalElementManager that is used by TemporalElementStore. TemporalElementStore provides factory methods to create various TemporalElements (Listing 4.3). A factory method for indeterminate intervals creates the composition tree depicted in Figure 4.3c and returns it as AnchoredTemporalElement object.

The TemporalElementStore has a fixed internal structure that is common across all scenarios. All TemporalElements are stored in the node table of a ParentChildGraph, which has six columns as defined in Table 4.1. For example, an Instant is stored as a row with the inf (infimum), sup (supremum), granularity id, and granularity context id of the Granule linked to the Instant. Additionally it stores a unique identifier of the

Listing 4.2: Excerpt of ParentChildNode with accessor methods for directed acyclic graph relationships.

```
public class ParentChildNode extends TableNode {
   public int getParentCount() {
      return super.m_graph.getOutDegree(this);
   }

   public ParentChildNode getParent(int idx) {
      int c = getGraph().getParentRow(m_row, idx);
      return (ParentChildNode) (c < 0 ? null : m_graph.getNode(c));
   }

   @SuppressWarnings("unchecked")
   public Iterator<? extends ParentChildNode> parents() {
      return super.outNeighbors();
   }
}
```

Listing 4.3: Factory methods in TemporalElementStore (selection).

```
public GenericTemporalElement addTemporalElement(long id, long inf, long
    sup, int granularityId, int granularityContextId, int kind) {...}
public GenericTemporalElement[] addTemporalElements(int nTuples, int kind)
    {...}
public Instant addInstant(long inf, long sup, Granularity granularity)
    {...}
public Instant addInstant(Granule granule) throws TemporalDataException
    {...}
public Span addSpan(long length, int granularityId) {...}
public Interval addInterval(Instant begin, Span span) throws
    TemporalDataException {...}
public AnchoredTemporalElement addIndeterminateInterval(Interval begin,
    Span maxLength, Span minLength, Interval end) {...}
public AnchoredTemporalElement addAnchoredSet(TemporalElement... elements)
    throws TemporalDataException {...}
```

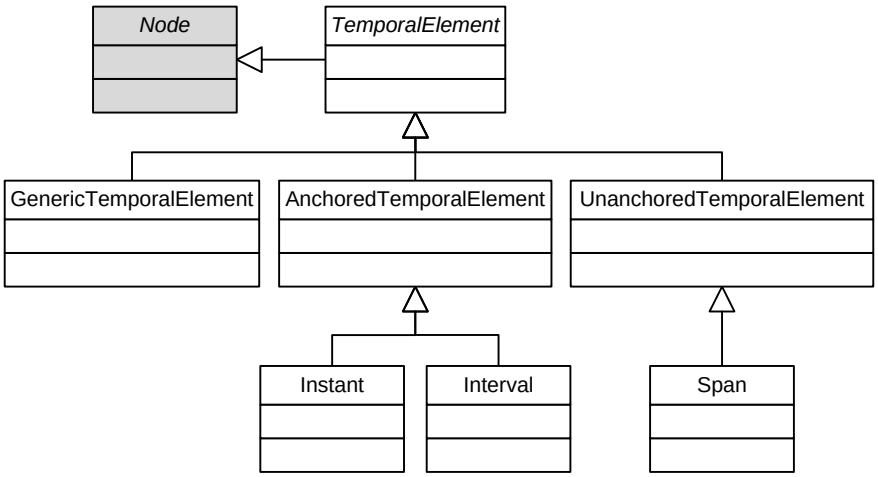


Figure 4.5.: Inheritance tree of the TemporalElement class. TimeBench provides subclasses for different temporal primitives.

TemporalElement and its kind (e.g., 2 := instant).

As shown in Listing 4.4, an Instant can be created from a Granule. These Granule objects are cached in a dedicated data structure within TemporalElementStore, while the infimum and supremum are copied into the relational data table. For efficient lookup, the infimum and the supremum are stored in the corresponding column for all AnchoredTemporalElements like Instant and Interval. For UnanchoredTemporalElements the infimum and the supremum are undefined and the columns inf and sup are instead used to store the length in granules. This is a minor architectural inconsistency, which was taken into account because the underlying prefuse Table does not efficiently handle sparse data columns.

4.3.4. Temporal Objects

The data items in a TemporalDataset are made available as TemporalObjects proxy tuples. These objects and the data structure provide accessor methods connecting them to TemporalElements and with each other in a ParentChildGraph (Listing 4.5). For application-specific data, the schema of the TemporalDataset can be extended with data columns as illustrated in Listing 4.6. Alternatively, the classes could be extended for an application-specific proxy tuple type.

Index structures are included to improve lookup efficiency. An interval index [Cormen et al., 2001] provides efficient queries for TemporalObject occurring in a given time

Listing 4.4: Illustration how TemporalElements can be created.

```
Granule g1, g2;
TemporalElementStore store;
Instant i1 = store.addInstant(g1);
Instant i2 = store.addInstant(g2);
Interval n = store.addInterval(i1, i2);
Span s = store.addSpan(42, i1.getGranularityId());

GenericTemporalElement gen = n.asGeneric();
boolean b = i1.getInf() == g1.getInf(); // true
b = n.getInf() == g1.getInf()); // true
```

Listing 4.5: Accessor methods of TemporalDataset and TemporalObject (selection).

```
public class TemporalDataset extends ParentChildGraph {
  public Iterable<TemporalObject> temporalObjects() {...}
  public Iterable<TemporalObject> temporalObjects(Predicate filter) {...}
  public Iterable<TemporalObject> getTemporalObjectsByElementId(long
      temporalId) {...}
}

public class TemporalObject extends ParentChildNode {
  public TemporalDataset getTemporalDataset() {...}
  public TemporalElement getTemporalElement() {...}
  public Iterable<TemporalObject> childObjects() {...}
  public Iterable<TemporalObject> parentObjects() {...}
}
```

Listing 4.6: Illustration how a TemporalDataset and one TemporalObject are created.

```
TemporalDataset tmpds = new TemporalDataset();
tmpds.addDataColumn("glucose", double.class, 0.0d);

TemporalObject obj = tmpds.addTemporalObject(instant1);
obj.set("glucose", 90.0);
```

Table 4.1.: Columns of the TemporalElement Table [Rind et al., 2013, p. 2251].

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)

interval based on the inf and sup columns in TemporalDataset. In particular, all possible qualitative relations [Allen, 1983] between the requested temporal objects and a given time point or interval. The interval index was initially implemented by Bilal Alsallakh and is based on a red-black tree, that has $O(log\ n)$ algorithmic complexity for temporal queries and for adding respectively removing elements to the index [Cormen et al., 2001]. TimeBench takes care to update the index automatically upon changes in the temporal elements. Furthermore, the id column in TemporalDataset, the id column in TemporalElementStore, and the TemporalElement's id in TemporalDataset are indexed using red-black trees implemented within prefuse.

TemporalDataset with temporal items connected by non-temporal edges is the default data structure provided by TimeBench, but other structures are possible. TemporalTable is a more generic alternative to TemporalDataset. It is a flat prefuse Table that may have more than one TemporalColumns. On this basis, it is possible to create a table with valid time and transaction time [Jensen et al., 1998], a graph with temporal edges, or a temporal tree.

4.3.5. Comparison to Aigner [2006]

While TimeBench's design started from the conceptual foundations laid by Aigner [2006, Chapter 10], the developers reconsidered the conceptual model in light of the desiderata and refined the architecture based on software design patterns but also on constraints from the programming language Java and the base framework prefuse. The most important differences are:

- The TemporalObject class stores its data aspects not in a dedicated DataElement class but via the attributes managed by its base class Tuple. This inheritance allows to visually represent the data aspects in a prefuse Visualization without any modifications specific to TimeBench.
- TemporalObjects and TemporalElements are not in an 1:n but in a m:n relationship with their child objects. For example, the same instant object could be the end of two interval objects (cp. e_5 in Figure 4.3b). While Aigner [2006] modeled a tree structure for these classes, TimeBench stores them in subclasses of ParentChildGraph, which is a directed graph with accessor methods to reflect parent and child relationships.
- The offset of TemporalElements was discarded because it would add more complexity than benefits.
- A GenericTemporalElements was implemented to allow more direct manipulation of temporal aspects.

4.4. Import/Export Functions

While the code listings above demonstrated how TemporalObjects can be created with Java code, a majority of usage scenarios requires that temporal data is loaded from an external data source. TimeBench provides import functions for text files with comma-separated values (Section 4.4.1) and calendars in iCalendar (Section 4.4.2). Furthermore it can import and export time series data from the R Project (Section 4.4.3).

Additionally a data exchange format is needed that is capable to express the structural details of the TemporalDataset so that it can be exported and re-imported without loss of information. Such an exchange format was built on top of GraphML (Section 4.4.4).

4.4.1. Comma-Separated Values

Text files with comma-separated values (CSV) are a frequently used data source in visual analytics prototyping. However, there are many variations of the file formats such as other delimiters instead of comma or different language-specific date formats. Furthermore, it is necessary to specify time abstraction aspects such as the granularity or the composition of TemporalElements. While the former could be substituted by a preceding data wrangling step (e.g., [Kandel et al., 2011]), the latter requires time-specific metadata about the data source.

The TimeBench class TextTableTemporalDatasetReader is capable to import a good selection of temporal data from CSV. By default, it looks for the first column, for which each value can be parsed as a Date object and creates an Instant primitive in the milliseconds granularity from these Dates. All other columns are made available as data columns of the TemporalDataset. For example, the CSV file shown in Listing B.1 can be imported without metadata because the date is given in a standardized format.

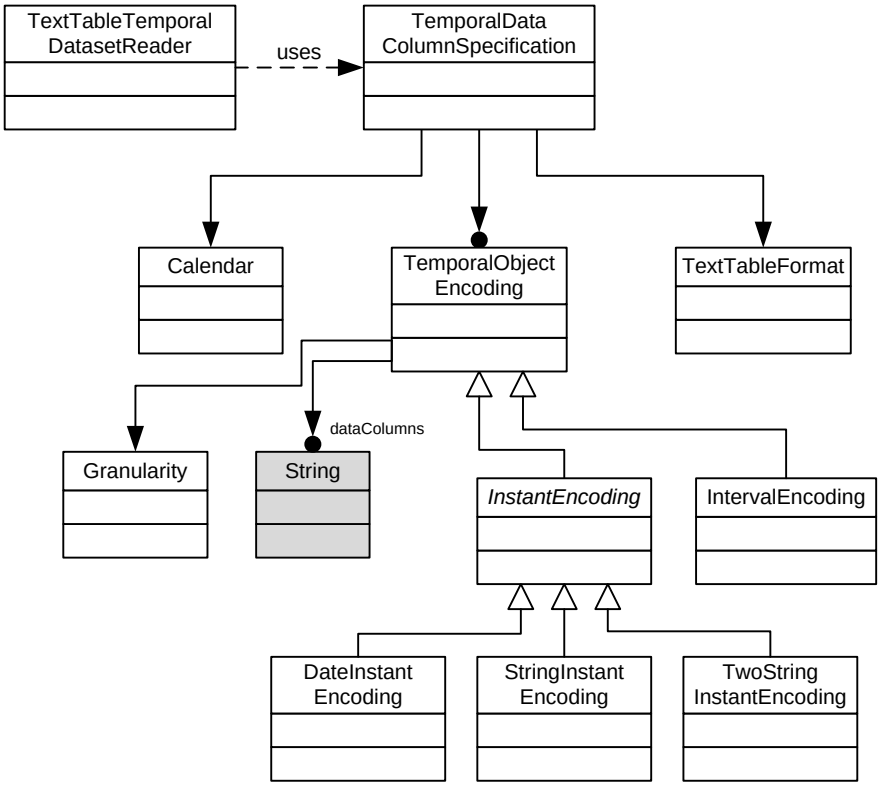


Figure 4.6.: Metadata structure for importing temporal data from CSV text files by the TextTableTemporalDatasetReader: A TemporalObject is created for each line of the input file and each TemporalObjectEncoding with the given data columns. The type of TemporalElement created depends on the subclass of TemporalObjectEncoding. Currently, TimeBench provides three encodings for Instant primitives and one for Interval primitives. All of them allow further configuration.
The diagram depicts classes and their relations using the extended object-modeling technique [Gamma et al., 1995; Heer and Agrawala, 2006]. The diagram is simplified and does not show all details.

To import data with regard of particular time abstractions or a varied file format, TextTableTemporalDatasetReader can be configured using metadata in TemporalData-

ColumnSpecification object. This metadata object is typically loaded from an XML file using the Java Architecture for XML Binding (JAXB) framework [Fialli and Vajjhala, 2006]. Figure 4.6 shows its data structure in the extended object-modeling technique and Appendix A lists the XML Schema. The metadata is specified as follows:

- One or more TemporalObjects are imported from each line of the CSV file depending on how many TemporalObjectEncoding are specified. Each TemporalObjectEncoding imports different columns for the time primitive and as data columns.
- How a time primitive is constructed from column values depends on the subclass of TemporalObjectEncoding. Currently there are four encodings: (1) DateInstantEncoding converts a automatically parsed Date object to an Instant of the given Granularity. (2) StringInstantEncoding additionally specifies a datetime format pattern for String to Date conversion. (3) TwoStringInstantEncoding builds a Date object from a date column and a time column. It provides two format patterns. (4) IntervalEncoding builds an Interval primitive from two Instant primitives that are previously imported by one of the other encodings.
- The Calendar is specified at the top level; the Granularity for each TemporalObjectEncoding.
- TextTableFormat specifies the delimiter and whether dates should be automatically parsed using the underlying prefuse code. This parsing offer less control over datetime format than TimeBench's metadata and is subject to the language settings at runtime.

Appendix B demonstrates one XML specification to import Instants from a non-standard time format (Listing B.4) and another XML specification to import Intervals (Listing B.5).

4.4.2. iCalendar

The class ICalenderTemporalDatasetReader provides an import function for calendars in iCalendar format. It has been developed by David Bauer during an internship with support from the author.

4.4.3. R Project

The class RConnector provides bidirectional data exchange between TimeBench and the R Project for Statistical Computing (R) [R Core Team, 2013]. In particular, it supports the R classes ts and zoo. `ts` is a very simple class supporting only regularly sampled time series. It is possible to time series in the Granularitys month, year, and decade and

to export a time series in the Granularity month. With zoo irregularly sampled Instants at the milliseconds Granularity can be imported and exported. The R class POSIXct stores the temporal data.

4.4.4. GraphML as Exchange Format

The class GraphMLTemporalDatasetWriter can save a TemporalDataset into a GraphML [Brandes et al., nd] file that maintains its internal graph structure. The class GraphMLTemporalDatasetReader[2] can load the GraphML file and recreate a TemporalDataset with the same data in the same structure. GraphML [Brandes et al., nd] is based on XML and its elements and attributes are defined by an XML Schema. This file format can be used to exchange data between different Visual Analytics solutions based on TimeBench. Since the file output conforms to the established GraphML file format, it can also be used for data exchange with graph tools such as Gephi [Bastian et al., 2009].

In order to be saved as a GraphML graph, the TemporalDataset and its associated TemporalElementStore, which are each structured as a directed acyclic graph, are transformed into one directed acyclic graph. This transformation needs to follows several conventions, so that it can be reversed:

- The GraphML id is build from the TemporalObject id prefixed with "o" or the TemporalElement id prefixed with "e". Other prefixes are not allowed.
- TemporalElements use the GraphML attributes _inf, _sup, _granularityID, _granularityContextID and _kind to store their internal data (cp. Table 4.1).
- TemporalObjects use GraphML attributes to store the data of their data columns. Neither in the TemporalDataset nor in its GraphML transformation is a data column name allowed that starts with an underscore.
- Each TemporalObject node must have exactly one directed edge to a TemporalElement node that represents their temporal relation.
- Directed edges between two TemporalObject nodes or between two TemporalElement nodes represent composition (from child to parent) like in the internally used ParentChildGraph.

The conventions guarantee that each GraphML file produced by TimeBench is valid with respect to the GraphML XML Schema. Conversely, each valid GraphML file that follows these convention can be loaded as a TemporalDataset. Listing 4.7 shows a

[2] GraphMLTemporalDatasetReader had initially been developed by Sascha Plessberger during an internship with support from the author and later refined by the author.

Listing 4.7: GraphML representation of a TemporalObject that records a blood sugar value (glucose = 90.0) at an Instant (the day June 8, 2011).

```
<graph id="temporalData" edgedefault="directed">
  <node id="t2">
    <data key="_inf">1307491200000</data>
    <data key="_sup">1307577599999</data>
    <data key="_granularityID">4</data>
    <data key="_granularityContextID">32767</data>
    <data key="_kind">2</data>
  </node>
  <node id="o0">
    <data key="glucose">90.0</data>
  </node>
  <edge source="o0" target="t2" />
</graph>
```

minimal example of one TemporalObject and one Instant represented as two GraphML nodes and one edge. A complete GraphML file with prologue and the declaration of GraphML attributes can be found in Appendix C.

However, the generic GraphML format has the limitation that a GraphML attribute can only contain a number, a boolean, or a string. While this is sufficient for some Visual Analytics scenarios, other scenarios would require TemporalDatasets with a data columns that stores a complex internal structure such as a linked list, a tree, or a map or a data column that stores an application-specific data structure. There are two possible workarounds: First, the data structure could be transformed into a string representation. Second, the GraphML schema could be extended to support this data structure.

4.5. Automated Analysis Operations

TimeBench's data structures can be transformed, aggregated, or enriched by automated analysis operations that are configured by user interaction. These operations can be constructed based on either the Operator design pattern or the Expression design pattern [Heer and Agrawala, 2006]. For both types of automated analysis operations, TimeBench provides some core operations and can be extended further.

Operator The Operator pattern allows to *"decompose visual data processing into a series of composable operators, enabling flexible and reconfigurable visual mappings"* [Heer and Agrawala, 2006, p. 857]. In TimeBench it is realized by subclasses of the the

prefuse class Action. While prefuse applies Actions only at the transition from abstract data (e.g., Graph) to visually-enriched data (e.g., VisualGraph), it was to possible employ Actions also to transform one instance of abstract data into another.

The InterpolationIndexingAction is a novel operator that transforms the values of a data column into values relative to an indexing point. Indexing is useful to compare different time series at different value ranges [Aigner et al., 2011a] and is often used together with horizon graphs [Reijner, 2008]. The operator uses TimeBench's data structure and internal indices to efficiently lookup TemporalObjects near the indexing point and interpolates the value.

The class ColumnToRowsTemporalDataTransformation is an operator that transforms a pivot table into a sequence of key/value pairs. This transformation is frequently needed when working with public datasets.

The GranularityAggregationAction is an operator that is particularly based on the time abstraction aspects supported by TimeBench.[3] It is configured with a sequence of Granularitys and rearranges the TemporalObjects of the input data set into the terminal nodes of a tree. The branches and non-terminals nodes are built based on granules of the configured Granularitys. Quantitative data of the TemporalObjects is aggregated based on a configurable aggregation method such as mean or sum. For example, from a daily time series of energy consumption could be transformed into a tree with a root TemporalObject that has the overall sum of energy consumed as a value, its children could be TemporalObjects for each month with the monthly sums of consumption, and their children would be TemporalObjects for the days of the month, which are copied from the original time series. The GranularityAggregationAction can be applied for many scenarios involving temporal aggregation, for example for the visualization technique GROOVE [Lammarsch et al., 2009].

Furthermore, an operator can be built using the RConnector to transform the data by algorithms implemented in a R package.

Expression This pattern *"provide[s] an expression language for data processing tasks such as specifying queries and computing derived values"* [Heer and Agrawala, 2006, p. 856]. TimeBench extends the expression language of prefuse with various expressions that leverage concepts of TimeBench's data structure and calendars. For example, with the AnchoredPredicate one can filter all the dataset to only include anchored

[3] GranularityAggregationAction was designed and implemented by Tim Lammarsch with support from the author.

TemporalElements. Other predicates allow filtering by primitive type, granularity, and context granularity.

4.6. Visual Representation and Interaction Techniques

Since TimeBench extends the polylithic framework prefuse, there are multiple possible approaches for interactive visual representation of a TemporalDataset.

- The TemporalDataset is an extension of the prefuse Graph. Therefore it can be added to any prefuse visualization either as a Graph or as the Table of nodes. Likewise, data aspects are available as attributes of the nodes using generic prefuse mechanisms. As mentioned above, a TemporalDataset with two data columns for glucose and cholesterol could be represented in a scatter plot. For this, the prefuse ShapeRenderer and two instances of the prefuse class AxisLayout would be used. A details-on-demand interaction for the scatter plot could be provided using the prefuse class ToolTipControl. After loading the TemporalDataset, no TimeBench code would be necessary.

- The TimeBench operator TimeAxisLayout lays out marks for TemporalObjects along a linear time axis. The class IntervalAxisLayout, which is a subclass of TimeAxisLayout, not only positions the mark for a TemporalObject but also sets its size to correspond with its temporal extent. These layouts are bundled with a group of classes that provide gridlines and interaction, which are based on the TimeVis framework by Peter Weishapl [2007] and earlier developments for the VisuExplore design study [Rind et al., 2011]. The class TimeScalePainter draws gridlines and axis labels using Granularity-based time units so that they fit to the current zoom level and screen resolution.

  To allow for interaction, an AdvancedTimeScale object needs to be set to the earliest and latest possible time and the prefuse class Display needs to be substituted with its subclass TimeAxisDisplay. This TimeAxisDisplay can be interactively panned and zoomed either by dragging the mouse (RangePanControl and RangeZoomControl) or by pushing toolbar buttons (RangePanAction and RangeZoomAction). A MouseTracker shows a support line and the timepoint at the current mouse position.

  Figure 4.7 illustrates TimeAxisLayout and its bundled functionality in a simple dot plot demo. To illustrate how these classes are embedded, the Java source code of this demo can be found in Appendix D.

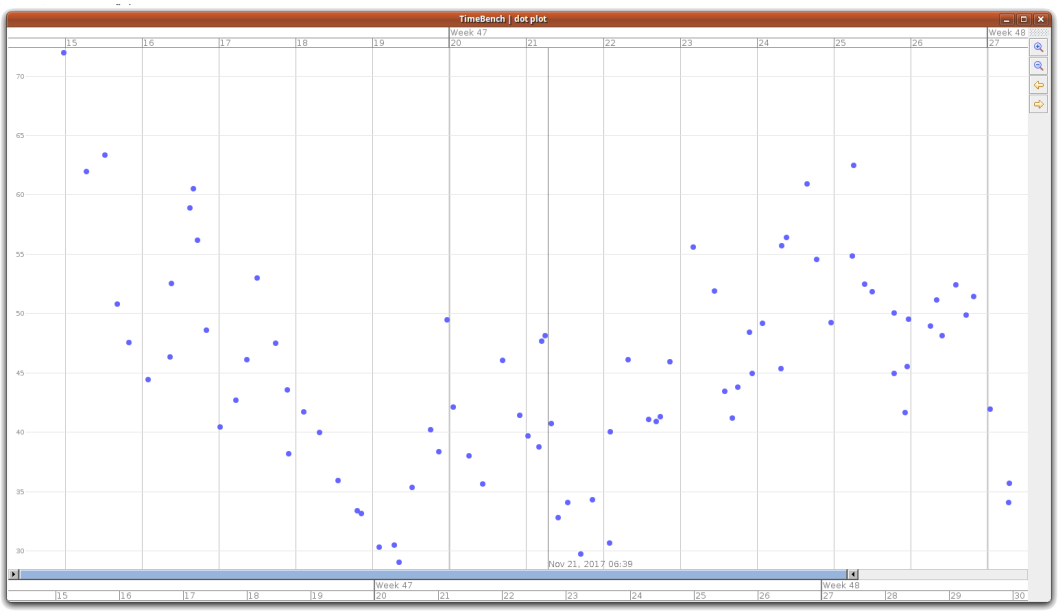


Figure 4.7.: Dot plot implemented with TimeAxisLayout

- The operators and expressions presented above can be used to transform the TemporalDataset into another data structure that lends itself better for visual representation and interaction.
- Finally, it is possible to create custom operators that are specific to TimeBench's data structure. For example, ThemeRiverLayout turns a TemporalDataset comprised of several time series into a group of polygons that will be rendered as a stacked area chart [Havre et al., 2002].

4.7. Summary

Together, the arrangement of the concepts and classes presented above results in TimeBench, a software framework for Visual Analytics of time-oriented data. It allows for a large degree of expressiveness of its calendar package, its data structure, and its set of operations in extension of prefuse. At the same time, it aims for comprehensibility based on its conceptual design.

TimeBench is free and open-source software under a BSD 2-Clause license and available at GitHub[4] It is implemented in the Java programming language, version 1.6 and

[4] https://github.com/ieg-vienna/TimeBench, last accessed Oct 29, 2017

extends the visualization framework prefuse [Heer et al., 2005]. It has the additional dependencies of Apache Commons Lang 3.0,[5] Apache log4j 1.2,[6] iCal4j 1.0.4,[7] and of the Java/R Interface (JRI), which is part of rJava.[8]

[5] https://commons.apache.org/proper/commons-lang/, last accessed Nov 2, 2017
[6] https://logging.apache.org/log4j/1.2/, last accessed Nov 2, 2017
[7] https://github.com/ical4j/ical4j, last accessed Nov 2, 2017
[8] https://www.rforge.net/rJava/, last accessed Nov 2, 2017

5 Evaluation

Development with a Visual Analytics software framework like TimeBench has some of the same characteristics like data analysis with an application domain-specific Visual Analytics solution: First, the number of possible users – developers respectively analysts with domain expertise – is much lower than for a usability study on an consumer e-commerce website. Second, their work typically lasts over longer stretches of time and involves many intertwined steps. These characteristics limit the repertoire of evaluation methods, because it is hard to find representative tasks for a laboratory study. Therefore, qualitative results inspection [Isenberg et al., 2013] is a widely used evaluation method in Visual Analytics. This inspection demonstrates that the solution is capable to achieve the expected results but does not test with users. In this spirit, Section 5.1 presents how TimeBench can be used to replicate two complex visualization techniques for time-oriented data. Another popular evaluation method is the case study with a small number of users in a realistic field setting [Shneiderman and Plaisant, 2006]. Section 5.2 presents two case studies of development projects that applied TimeBench. In addition, Section 5.3 surveys community adoption of TimeBench.

5.1. Demonstrations of Expressiveness

The software repository for TimeBench includes a collection of demonstration applications. On the one hand, these applications showcase the expressiveness of TimeBench.

On the other hand, they can serve as training material for developers starting to use TimeBench. Next, two of them will be introduced in more detail.

5.1.1. Horizon Graph

The horizon graph is a space-efficient visualization technique for multi-variate time series. It was introduced by Reijner [2008] as a variant of two-tone pseudo coloring [Saito et al., 2005] and has received considerable attention in the visualization community [Heer et al., 2009; Javed et al., 2010; Perin et al., 2013; Federico et al., 2014].

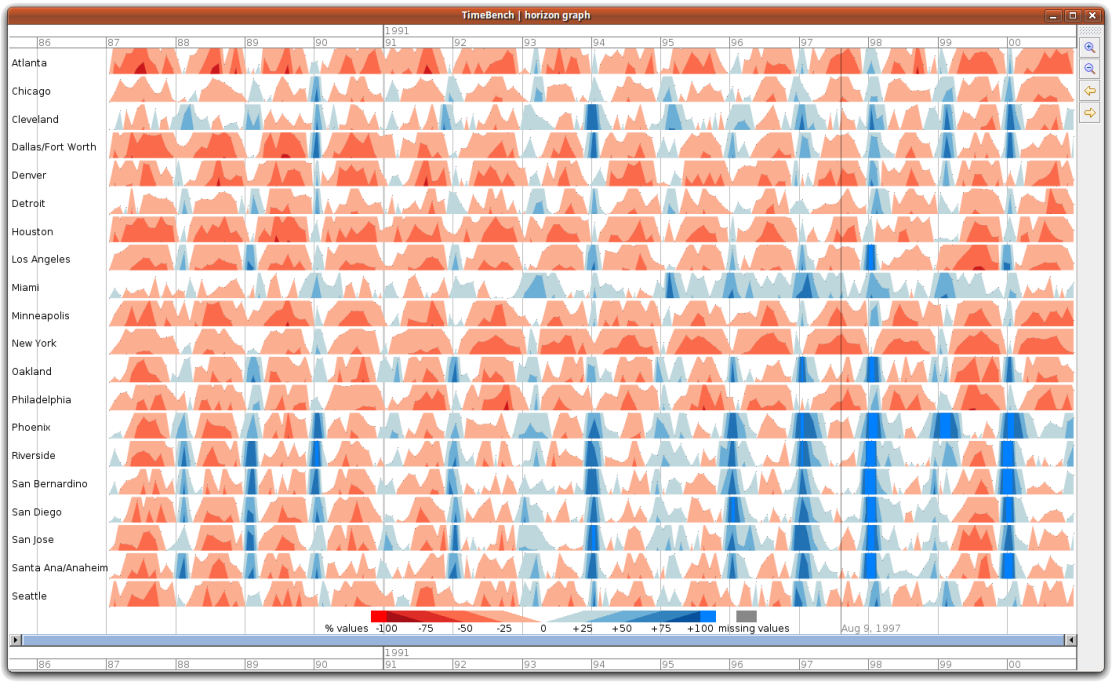


Figure 5.1.: Horizon graph implemented with TimeBench

Figure 5.1 shows the TimeBench implementation of the horizon graph technique. It was built as follows: The data are 20 time series with a length of 168 months and contain public health about 20 cities data from the NMMAPS study [Peng and Welty, 2004]. First, the data is loaded from the CSV file as Instants of the Granularity month. Then it is transformed from a pivot table to city/value pairs. The class HorizonGraphAction and some other specialized operators transform the time series into colored polygons. The horizontal position of the polygon's control points is controlled by a TimeAxisLayout. Finally, the rendered polygons result in a horizon graph. Additionally, control points that are represent a months are rendered as a small grey dot. Since, the cities' values are

in different value ranges, the InterpolationIndexingAction transforms them to relative values. An IndexingControl allows the use to select a month as the indexing point by clicking on one of its grey control points. Furthermore, the TimeAxisLayout allows for interactive zooming and panning.

5.1.2. PlanningLines

The PlanningLines technique was devised by Aigner et al. [2005] to represent temporal indeterminacy in clinical guidelines and protocols. It extends the notation of a Gantt chart by anchoring tasks to indeterminate intervals that are visually represented using the glyph depicted in Figure 3.3.

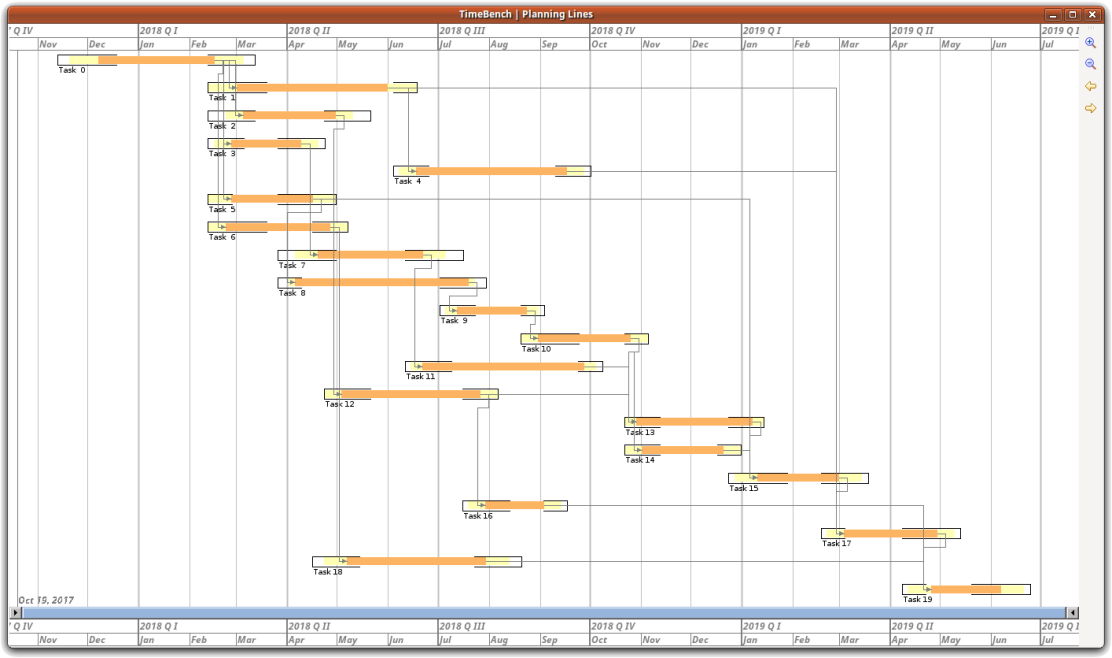


Figure 5.2.: PlanningLines implemented with TimeBench

Figure 5.2 depicts the TimeBench implementation of the PlanningLines technique. It was realized as follows: First, 20 tasks of random indeterminate duration are created and scheduled using a simple forward planning algorithm. Dependencies between tasks are represented as directed edges between TemporalObjects and are shown as thin black lines. For each TemporalObject five marks will be created that are organized by prefuse in different groups: the minimum and the maximum duration will be rendered as rectangles of different color, the start and the end interval will be rendered as

black polygons, and the tasks title will be displayed as a label below the glyph. Four instances of IntervalAxisLayout set the x-position and width of the glyph parts. For this, the IntervalAxisLayout can be configured to use a specific child primitive its layout. For example, the layout for the black polygon mark representing the start interval is calculated based on the first child of the AnchoredTemporalElement associated to the TemporalObject. The task labels are positioned relative to this mark of the start interval. Since all IntervalAxisLayout instances share the same AdvancedTimeScale, the PlanningLines chart can be zoomed and panned interactively.

5.2. Case Studies of Usage in Development Projects

To investigate how TimeBench can be deployed in the realistic setting of Visual Analytics prototyping, another Master student and a group of two high school students were invited to base the implementation of their final theses on TimeBench.

5.2.1. Model Selection in Time Series Analysis

TiMoVA [Bögl, 2013; Bögl et al., 2013] is visual analytics solution for model selection in time series analysis (Figure 5.3). It employs TimeBench's CSV import to load time series data from text files and displays them in a zoomable line plot (top left). This line plot can also be used to select a segment of the time series for analysis. Time series analysis and other statistical computations are performed using the R Project for Statistical Computing (R) [R Core Team, 2013]. The data exchange between R and the Java-based visual interface uses TimeBench's RConnector. In particular, a seasonal ARIMA model [Shumway and Stoffer, 2011] is fitted to the time series and complexity of the plots can be interactively adjusted using the plots in the bottom left. Diagnostic plots in the right half of the visual interface allow the inspection of the residuals (i.e., difference between the data and the model's output). An extension of TiMoVA [Bögl et al., 2015] includes the prediction of time series values as an additional way to assess the quality of a time series model.

TiMoVA was developed within the scope of M.Sc. thesis in Medical Informatics. The student had no prior experience with TimeBench or prefuse. He could reuse code from the demonstration applications for the import from text files and the interactive line plot based on TimeAxisLayout. Instead of reimplementing these standard features, he could focus on making statistical methods smoothly accessible in an interactive visual interface. Thus, he successfully employed TimeBench without needing to concern himself with details of its data structures or calendars.

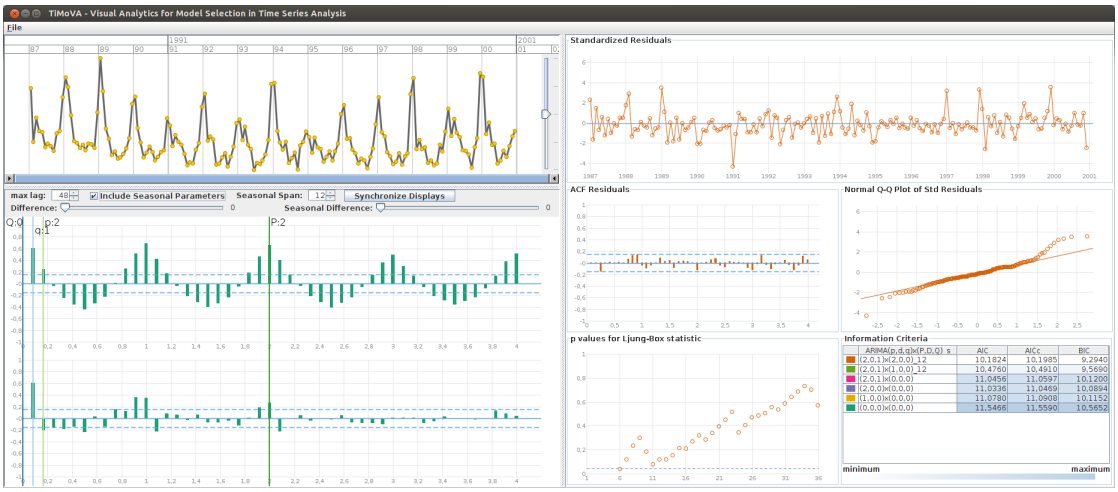


Figure 5.3.: TiMoVA prototype [Bögl et al., 2013]: It loads and visualizes time series based on TimeBench and connects to R for time series analysis. © *2013 IEEE*

5.2.2. Exploration of Irregularly Sampled Data

If time-oriented data is irregularly distributed over a large portion of time, its exploration gets more complicated as data are more dense in some parts while other stretches of time are empty without any data. To improve the user experience, two variants of a linear time axis mapping were developed [Bauer and Pleßberger, 2013]: For the first variant, the mapping of between time and display space is distorted based on the density of data, which is computed using the `GranularityAggregationAction`. The second variant compresses the time scale at gaps without any data that are longer than a threshold. The user can interactively configure both variants by setting the minimum gaps length and strength of distortion.

This work was conducted as the final project of two high school students specialized in information technology. The student were involved in the development of TimeBench during a one-month internship with the author's research group. Before that, they had no experience of Java or prefuse because their curriculum focused on C#. For their school they reported a total development time of 165 person hours (h), 10 h for setup on basis of a demo, 109 h for the distorted time scale, and 46 h for the time scale with gaps. Within this project, the students succeeded in independently extending components of TimeBench.

5.3. Community Adoption

An article about the TimeBench framework was published in the IEEE Transactions on Visualization and Computer Graphics [Rind et al., 2013]. As of completion of this thesis, this article has 23 citations in Google Scholar.[1] Besides the case studies presented above, at least two published projects applied TimeBench: Lammarsch et al. [2014] developed a novel visual analytics approach for pattern discovery based on temporal predicates between events of interest. [Amor-Amorós et al., 2017] used it in a design study to support aerial image georeferencing in order to identify unexploded bombs.

5.4. Summary

While the adoption of TimeBench by more projects could be desired, the demonstrations and case studies show that TimeBench is a capable basis for complex Visual Analytics solutions and provides good developer accessibility.

[1] https://scholar.google.at/scholar?cites=14703250657204117914&as_sdt=2005&sciodt=0,5&hl=de, last accessed Nov 12, 2017

6 Discussion

The design decisions leading to the software architecture and prototypical implementation of TimeBench were guided by the desiderata established in Chapter 3. The previous chapter reported collected evidence about how well TimeBench addresses these desiderata. The fulfillment of desiderata will be discussed next. Afterwards, some aspects of TimeBench will be critically reflected.

6.1. Fulfillment of Desiderata

TimeBench aims to address three desiderata on expressiveness regarding time abstraction and three desiderata on efficiency for software development. The desiderata were prioritized in order listed below with expressiveness having a higher priority than efficiency.

Granularity & calendars The calendar package of TimeBench provides a unified programming interface for different calendars and different calendar backends. It provides classes for Granules and Granularitys and can express particular granules such as 'the day November 2, 2017', general granules such as 'the first day of a week', or general granules in context such as 'every Monday in October 2017'. Granules are directly connected to the data structures as for example an Instants can be create from a Granule. Furthermore, the calendar package provides the basis for human-readable axis labels in the widely used TimeAxisLayout and for

the GranularityAggregationAction, which was used for example in the case study to distort the display of irregularly sampled data (Section 5.2.2).

Time primitives The class TemporalElementStore provides an abstraction layer between application data associated to TemporalObjects and time primitives. This makes it possible to support the three widely used primitives Instant, Interval, and Span. It also allows the definition of custom primitives that can be composed of a hierarchy of primitives. TimeBench provides an object-oriented programming interface for primitives with subclasses of TemporalElement for each type of primitive. TimeBench also provides an import function from text files with comma-separated values that supports data anchored at Instants, Intervals, or custom primitives by extension. The GraphML example listed in Appendix C illustrates a heterogeneous time-oriented data set with different primitives in a fictitious healthcare scenario. Furthermore, the PlanningLines example demonstrates the extensibility of time primitives by implementing the indeterminate interval as a composite of two Intervals and two Spans (Section 5.1.2).

Determinacy While determinacy of temporal data is a standard feature of many software frameworks, TimeBench additionally supports three forms of indeterminacy: (1) The indeterminate intervals implemented in the PlanningLines example illustrate how TimeBench can capture indeterminacy explicitly as a time primitives. (2) TimeBench anchors temporal data at the precision of a Granule. For example, a TemporalObject can be anchored at an Instant of Granularity week, whereas a simpler data structure might require an artificial milliseconds precision. (3) The general granules can specify partly determined temporal information such as 'on a Monday' without fixing 'which Monday'.

Common data structure TimeBench is constructed around a the data structure presented in Section 4.3. The same underlying structure could be used for all the examples presented in this work and the demonstration applications packaged with the source code. This common data structure is enabled by relational data tables that can be accessed via extensible classes that act as Proxy Tuples [Heer and Agrawala, 2006].

Developer accessibility While TimeBench needs to introduce some complexity in order to support the desiderata listed above, its architecture was designed with a focus on developer accessibility: (1) The data structures are built based on a simple conceptual model described in Section 4.3.1. (2) The Proxy Tuple design pattern [Heer and Agrawala, 2006] allows access to temporal data in an object-oriented manner. (3) Calendar operations, time primitives, and datasets are built

as three nested layers. Developers of a new TimeBench component will typically only need to manipulate objects from one or two layers to build the needed functionality. Developer using existing TimeBench components will typically not be concerned with internal layers. This is illustrated well by the two case studies presented in Section 5.2. (4) TimeBench does not introduce a completely new software framework but extends prefuse [Heer et al., 2005]. (5) The demonstration applications packaged with the source code provide an novice TimeBench developers a quick start to learn and experiment with its components.

Runtime efficiency Again, the constraints imposed by other desiderata limit the options for optimizing runtime efficiency. Within these limits, TimeBench builds it data structures on the well-tested relational data tables of prefuse [Heer et al., 2005] that employ the Data Column design pattern [Heer and Agrawala, 2006] for efficient storage. In addition, the temporal extent (infimum and supremum) of anchored time primitives such as Instants is stored in two columns of type long. Furthermore, the interval-based lookup of TemporalObjects and various other lookups are optimized using red-black trees [Cormen et al., 2001]. The runtime performance of TimeBench can be estimated by using the demonstration applications. For example, the horizon graph can be interactively zoomed and re-indexed smoothly with a data size of $20 \times 12 \times 14 = 3,360$ TemporalObjects.

In summary, it can be concluded that TimeBench addresses the desiderata well.

6.2. Critical Reflection

Beyond the desiderata discussed above, some questions arose during the development of TimeBench and will be critically reflected below.

Visual Analytics One could question whether TimeBench is a software framework for Visual Analytics or for Information Visualization. On the one hand, its base framework prefuse [Heer et al., 2005] and its related work are rather from known as Information Visualization frameworks. Furthermore, the majority of demonstration applications are reimplementations of classical Information Visualization techniques like horizon graph or stream graph. On the other hand, time plays a very important role in Visual Analytics, which is demonstrated by the dedicated chapter on "Space and Time" in the roadmap book edited by Keim et al. [2010]. TimeBench supports Visual Analytics with time-oriented data not only by information methods but it also intertwines them with automated analysis operations such as the GranularityAggregationAction (Section 4.5). The TiMoVA [Bögl, 2013; Bögl et al., 2013] case study (Section 5.2.1) and the

pattern discovery approach by Lammarsch et al. [2014] demonstrate how TimeBench could be effectively employed for Visual Analytics.

Java In recent years, Java has lost popularity as a development platform to JavaScript-based web applications. Current web browsers make it increasingly difficult to run code developed in Java for stability and security reasons – neither Java applets nor Java Web Start can be expected to be available by default. Due to these technical constraints and JavaScript being the primary programming language of many developers, D3.js [Bostock et al., 2011] is now one of the most popular frameworks for visualization prototyping. These devaluations of Java also affects TimeBench, which was primarily intended for rapid prototyping in Visual Analytics research.

However Java is still in use for rich client user interfaces such as Eclipse,[1] FreeMind,[2] Cytoscape [Shannon et al., 2003], or Gephi [Bastian et al., 2009]. Likewise, it is still used for Visual Analytics prototyping (e.g., [Janetzko et al., 2014; Wagner et al., 2017]).

In addition, web applications can interface via web services with Java code running on a server. Thus, a TimeBench-based time analytics server would be possible. Furthermore, given the increasing maturity of TypeScript,[3] a typed extension of JavaScript that compiles to JavaScript, reimplementing the conceptual design of TimeBench in JavaScript and linking it the feature-rich D3.js framework would be a promising direction.

[1] http://www.eclipse.org/, last accessed Oct 29, 2017
[2] http://freemind.sourceforge.net/wiki/index.php/Main_Page, last accessed Oct 29, 2017
[3] https://www.typescriptlang.org/, last accessed Oct 29, 2017

7 Conclusions

The starting point of this work was to better support Visual Analytics of time-oriented data. In particular, for rapid prototyping in research projects, a specialized software framework was needed. On the one hand, this framework should be expressive to support the time abstraction aspects of time-oriented data: granularity & calendars, time primitives, and determinacy [Aigner et al., 2011b]. On the other hand, it should be efficient by supporting heterogeneous time-oriented data with a common data structure, be accessible to developers, and efficient at runtime. Thus, this work tackled the main research question: *How can a software framework support Visual Analytics of time-oriented data in an expressive and efficient way?*

Answering this research question, this work proposes the software architecture and prototypical implementation of TimeBench. In particular, this work focuses on the data structures and import/export functions developed by its author.

Which data structure can manage time-oriented data that is linked to heterogeneous, hierarchically composed time primitives? TimeBench stores time-oriented data in two directed acyclic graphs, one for TemporalObjects with application-specific attributes and one for TemporalElements that capture the time abstraction aspects. Each TemporalObject is connected with exactly one TemporalElement, while a TemporalElement may have zero, one, or multiple connected TemporalObjects. Directed graph edges represent composition and allow, e.g., an Interval to be composed of two Instants. These data items can be manipulated using object-oriented programming, because TimeBench applies the Proxy Tuple design pattern [Heer and Agrawala, 2006] to encapsulate the

internal data structure. Internally, data are managed using the Data Column design pattern as implemented in a prefuse Table [Heer et al., 2005].

How can text files with comma-separated values be annotated for configurable import of time-oriented data? To get time-oriented data into TimeBench it provides a widely configurable importer from text files with comma-separated values. Custom datetime formats can be specified and both instants and intervals are supported. Its configuration can be loaded from an XML file. that is based on the XML Schema listed in Appendix A. Furthermore, it provides a calendar importer from the iCalendar format, bidirectional data exchange with two R packages for time series, and a GraphML-based data exchange format.

Integrating these features with a calendar package, automated analysis operations, and visual representation and interaction techniques, TimeBench results in an expressive and efficient framework for Visual Analytics development. For its evaluation, two complex visualization techniques were reimplemented as application examples. Furthermore, two case studies observed one student project that applied TimeBench and one student project that extended its visual representation support.

Future Work While TimeBench at its current state is a complete and useful software framework that has already been practically applied, there are ample opportunities for further research and development:

- The set of included operators for import/export, automated analysis operations, visual representation techniques, and interaction techniques could be increased. For example, an interactive spiral display [Weber et al., 2001] would support the investigation of seasonal patterns.
- A larger set of demonstration applications would increase developer accessibility. Furthermore, stepwise tutorials targeting different development needs would reduce hurdles for TimeBench novices even further.
- In addition to existing case studies and demonstration applications that each focus on one particular form of time-oriented data, large case study with heterogeneous data would demonstrate the added value of TimeBench more clearly. Such a case study could for example be based on healthcare data [Aigner et al., 2012; Rind et al., 2017].
- Finally, Lammarsch et al. [2011] envisioned an extended Visual Analytics framework that would extends TimeBench with hypothesis and models about time-oriented data. For this, additional metadata would be needed within the data structure. Furthermore, a set of operators are needed so that these data and metadata can be effectively applied.

All these development directions will contribute the Visual Analytics community by providing better software infrastructure. TimeBench now is a solid starting point for these endeavors.

Appendix

A

XML Schema for Importing CSV Data

Below, the complete XML schema for providing metadata to a text file with comma-separated values is listed. The metadata comprises details on the data format and on time abstractions and is described in detail in Section 4.4.1.

The XML schema was auto-generated from Java code and annotations using the Java Architecture for XML Binding (JAXB) framework [Fialli and Vajjhala, 2006].

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="temporal-data-column-specifaction"
      type="temporalDataColumnSpecification"/>

  <xs:complexType name="temporalDataColumnSpecification">
    <xs:sequence>
      <xs:element name="calendar" type="xs:int"/>
      <xs:element name="fail-on-illegal-rows" type="xs:boolean"/>
      <xs:element name="encodings" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
              <xs:element name="date-instant" type="dateInstantEncoding"/>
              <xs:element name="string-instant" type="stringInstantEncoding"/>
```

```
          <xs:element name="two-string-instant"
              type="twoStringInstantEncoding"/>
          <xs:element name="interval" type="intervalEncoding"/>
        </xs:choice>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="text-table" type="textTableFormat" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="dateInstantEncoding">
 <xs:complexContent>
  <xs:extension base="instantEncoding">
   <xs:sequence>
    <xs:element name="temporal-column" type="xs:string"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="instantEncoding" abstract="true">
 <xs:complexContent>
  <xs:extension base="temporalObjectEncoding">
   <xs:sequence/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="temporalObjectEncoding" abstract="true">
 <xs:sequence>
  <xs:element name="data-element" minOccurs="0">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="column" type="xs:string" minOccurs="0"
         maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="granularity-id" type="xs:int"/>
  <xs:element name="granularity-context-id" type="xs:int"/>
 </xs:sequence>
 <xs:attribute name="key" type="xs:string" use="required"/>
</xs:complexType>
```

```
<xs:complexType name="stringInstantEncoding">
  <xs:complexContent>
    <xs:extension base="instantEncoding">
      <xs:sequence>
        <xs:element name="temporal-column" type="xs:string"/>
        <xs:element name="date-time-pattern" type="xs:string" minOccurs="0"/>
        <xs:element name="language" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="twoStringInstantEncoding">
  <xs:complexContent>
    <xs:extension base="instantEncoding">
      <xs:sequence>
        <xs:element name="date-column" type="xs:string"/>
        <xs:element name="time-column" type="xs:string"/>
        <xs:element name="date-pattern" type="xs:string" minOccurs="0"/>
        <xs:element name="time-pattern" type="xs:string" minOccurs="0"/>
        <xs:element name="language" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="intervalEncoding">
  <xs:complexContent>
    <xs:extension base="temporalObjectEncoding">
      <xs:sequence>
        <xs:element name="begin" type="xs:string" minOccurs="0"/>
        <xs:element name="end" type="xs:string" minOccurs="0"/>
        <xs:element name="span" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="textTableFormat">
  <xs:sequence>
    <xs:element name="method" type="method"/>
    <xs:element name="has-header" type="xs:boolean"/>
    <xs:element name="parse-dates" type="xs:boolean"/>
```

```
      <xs:element name="delimiter-regex" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="method">
    <xs:restriction base="xs:string">
      <xs:enumeration value="csv"/>
      <xs:enumeration value="regex-delimited"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

B

Example Specifications for Importing CSV Data

The listings below demonstrate how TimeBench can import three example text files with comma-separated values containing instant (Listings B.1 and B.2) and interval data (Listing B.3). TextTableTemporalDatasetReader uses metadata specified as shown in Listing B.4 and Listing B.5. The metadata is described in detail in Section 4.4.1.

Listing B.1: Example CSV working in auto-detection mode.

```
date,value
1987-01-01,55
1987-01-02,73
1987-01-03,64
1987-01-04,57
1987-01-05,56
1987-01-06,65
1987-01-07,43
1987-01-08,69
```

Listing B.2: Example CSV with Instant data.

```
date,value
11.4.2005 9:00,30.0
11.4.2005 9:05,27
11.4.2005 9:10,42
11.4.2005 9:15,38
11.4.2005 9:20,39
11.4.2005 9:25,29
11.4.2005 9:30,31
11.4.2005 9:35,29
11.4.2005 9:40,31
11.4.2005 9:45,37
11.4.2005 9:50,19
11.4.2005 9:55,34
11.4.2005 10:00,26
11.4.2005 10:05,26
11.4.2005 10:10,26
11.4.2005 10:15,39
11.4.2005 10:20,29
11.4.2005 10:25,25
11.4.2005 10:30,32
```

Listing B.3: Example CSV with Interval data.

```
begin,end,event,location
2016-10-23,2016-10-28,IEEE VIS,Baltimore
2015-10-24,2015-10-30,IEEE VIS,Chicago
2014-11-09,2014-11-14,IEEE VIS,Paris
2016-06-06,2016-06-10,EuroVis,Groningen
2015-05-25,2015-05-29,EuroVis,Cagliari
2014-06-09,2014-06-13,EuroVis,Swansea
2017-04-18,2017-04-21,PacificVis,Seoul
2016-04-19,2016-04-22,PacificVis,Taipei
2015-04-14,2015-04-17,PacificVis,Hangzhou
2014-03-04,2014-03-07,PacificVis,Yokohama
```

Listing B.4: Example specification for importing the Instant data shown in Listing B.2.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<temporal-data-column-specifaction>
   <calendar>0</calendar>
   <fail-on-illegal-rows>true</fail-on-illegal-rows>
   <encodings>
      <string-instant key="date">
         <data-element>
            <column>value</column>
         </data-element>
         <granularity-id>0</granularity-id>
         <granularity-context-id>1111111111</granularity-context-id>
         <temporal-column>date</temporal-column>
         <date-time-pattern>d.M.yyyy H:mm</date-time-pattern>
         <language>de</language>
      </string-instant>
   </encodings>
   <text-table>
      <method>csv</method>
      <has-header>true</has-header>
      <parse-dates>false</parse-dates>
   </text-table>
</temporal-data-column-specifaction>
```

Listing B.5: Example specification for importing the Interval data shown in Listing B.3.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<temporal-data-column-specifaction>
   <calendar>0</calendar>
   <fail-on-illegal-rows>true</fail-on-illegal-rows>
   <encodings>
      <date-instant key="begin">
         <granularity-id>4</granularity-id>
         <granularity-context-id>1111111111</granularity-context-id>
         <temporal-column>begin</temporal-column>
      </date-instant>
      <date-instant key="end">
         <granularity-id>4</granularity-id>
         <granularity-context-id>1111111111</granularity-context-id>
         <temporal-column>end</temporal-column>
      </date-instant>
      <interval key="date">
         <data-element>
            <column>event</column>
            <column>location</column>
         </data-element>
         <begin>begin</begin>
         <end>end</end>
      </interval>
   </encodings>
   <text-table>
      <method>csv</method>
      <has-header>true</has-header>
      <parse-dates>true</parse-dates>
   </text-table>
</temporal-data-column-specifaction>
```

C
GraphML Example

The listing below shows a complete GraphML file (cp. Section 4.4.4) that stores a heterogeneous time-oriented dataset comprised of eight TemporalElements and seven TemporalObjects in a fictitious healthcare scenario. Their structure is also depicted in Figure 4.3d on a conceptual level.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
  xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
      http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <key id="_inf" attr.name="_inf" attr.type="long" for="node" />
  <key id="_sup" attr.name="_sup" attr.type="long" for="node" />
  <key id="_granularityID" attr.name="_granularityID" attr.type="int"
      for="node" />
  <key id="_granularityContextID" attr.name="_granularityContextID"
      attr.type="int" for="node" />
  <key id="_kind" attr.name="_kind" attr.type="int" for="node" />
  <key id="name" attr.name="name" attr.type="string" for="node" />
  <key id="value" attr.name="value" attr.type="double" for="node" />
  <graph id="temporalData" edgedefault="directed">
    <node id="t0">
      <data key="_inf">1304406702000</data>
      <data key="_sup">1304406702999</data>
```

```
    <data key="_granularityID">1</data>
    <data key="_granularityContextID">32767</data>
    <data key="_kind">2</data>
  </node>
  <node id="t1">
    <data key="_inf">1304294400000</data>
    <data key="_sup">1304380799999</data>
    <data key="_granularityID">4</data>
    <data key="_granularityContextID">32767</data>
    <data key="_kind">2</data>
  </node>
  <node id="t2">
    <data key="_inf">1307491200000</data>
    <data key="_sup">1307577599999</data>
    <data key="_granularityID">4</data>
    <data key="_granularityContextID">32767</data>
    <data key="_kind">2</data>
  </node>
  <node id="t3">
    <data key="_inf">1304294400000</data>
    <data key="_sup">1307577599999</data>
    <data key="_granularityID">4</data>
    <data key="_granularityContextID">32767</data>
    <data key="_kind">3</data>
  </node>
  <node id="t4">
    <data key="_inf">23</data>
    <data key="_sup">23</data>
    <data key="_granularityID">4</data>
    <data key="_granularityContextID">-1</data>
    <data key="_kind">0</data>
  </node>
  <node id="t5">
    <data key="_inf">1304294400000</data>
    <data key="_sup">1306281599999</data>
    <data key="_granularityID">4</data>
    <data key="_granularityContextID">32767</data>
    <data key="_kind">3</data>
  </node>
  <node id="t6">
    <data key="_inf">1304406702000</data>
    <data key="_sup">1307577599999</data>
    <data key="_granularityID">4</data>
    <data key="_granularityContextID">32767</data>
```

```
    <data key="_kind">1</data>
  </node>
  <node id="t7">
    <data key="_inf">1304294400000</data>
    <data key="_sup">1307577599999</data>
    <data key="_granularityID">4</data>
    <data key="_granularityContextID">32767</data>
    <data key="_kind">1</data>
  </node>
  <node id="o0">
    <data key="name">HbA1c</data>
    <data key="value">8.5</data>
  </node>
  <node id="o1">
    <data key="name">VZI</data>
    <data key="value">12.0</data>
  </node>
  <node id="o2">
    <data key="name">ALT</data>
    <data key="value">1.5</data>
  </node>
  <node id="o3">
    <data key="name">Misch</data>
    <data key="value">1.0</data>
  </node>
  <node id="o4">
    <data key="name">HbA1c</data>
    <data key="value">7.2</data>
  </node>
  <node id="o5">
    <data key="name">HbA1c</data>
    <data key="value">7.75</data>
  </node>
  <node id="o6">
    <data key="name">Insulin</data>
    <data key="value">1.0</data>
  </node>
  <edge source="o0" target="t0" />
  <edge source="o1" target="t3" />
  <edge source="o2" target="t5" />
  <edge source="o3" target="t5" />
  <edge source="o4" target="t2" />
  <edge source="o5" target="t6" />
  <edge source="o6" target="t7" />
```

```
    <edge source="o0" target="o5" />
    <edge source="o4" target="o5" />
    <edge source="o1" target="o6" />
    <edge source="o2" target="o6" />
    <edge source="o3" target="o6" />
    <edge source="t1" target="t3" />
    <edge source="t2" target="t3" />
    <edge source="t1" target="t5" />
    <edge source="t4" target="t5" />
    <edge source="t0" target="t6" />
    <edge source="t2" target="t6" />
    <edge source="t3" target="t7" />
    <edge source="t5" target="t7" />
  </graph>
</graphml>
```

D
Dot Plot Example

The listing below shows the complete source code of the DotPlotDemo. This demo is a very simple example of how TimeAxisLayout can be used.

```
package timeBench.demo.vis;

import ieg.prefuse.data.DataHelper;

import javax.swing.BorderFactory;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import prefuse.Constants;
import prefuse.Visualization;
import prefuse.action.ActionList;
import prefuse.action.RepaintAction;
import prefuse.action.assignment.ColorAction;
import prefuse.action.assignment.ShapeAction;
import prefuse.action.layout.AxisLabelLayout;
import prefuse.action.layout.AxisLayout;
import prefuse.controls.ToolTipControl;
import prefuse.render.AxisRenderer;
import prefuse.render.DefaultRendererFactory;
import prefuse.render.ShapeRenderer;
```

```
import prefuse.util.ColorLib;
import prefuse.visual.VisualItem;
import prefuse.visual.expression.InGroupPredicate;
import prefuse.visual.expression.VisiblePredicate;
import prefuse.visual.sort.ItemSorter;
import timeBench.action.layout.TimeAxisLayout;
import timeBench.action.layout.timescale.AdvancedTimeScale;
import timeBench.action.layout.timescale.RangeAdapter;
import timeBench.data.TemporalDataException;
import timeBench.data.TemporalDataset;
import timeBench.ui.TimeAxisDisplay;
import timeBench.util.DemoEnvironmentFactory;

/**
 * Simple demo of a dot plot showing a numerical variable over time.
 * @author Rind
 */
public class DotPlotDemo {

   private static final String COL_DATA = "value";

   private static final String GROUP_DATA = "data";
   private static final String GROUP_AXIS_LABELS = "ylab";

   /**
    * @param args
    * @throws TemporalDataException
    */
   public static void main(String[] args) throws TemporalDataException {
      TemporalDataset tmpds = DemoEnvironmentFactory
            .generateRandomNumericalInstantData(100, COL_DATA);
      DataHelper.printTable(System.out, tmpds.getNodeTable());

      final Visualization vis = new Visualization();
      final TimeAxisDisplay display = new TimeAxisDisplay(vis);
      // display width must be set before the time scale
      // otherwise the initial layout does not match the display width
      display.setSize(700, 450);

      // ------------------------------------------------------------------------
      // STEP 1: setup the visualized data & time scale
      vis.addTable(GROUP_DATA, tmpds.getTemporalObjectTable());

      long border = (tmpds.getSup() - tmpds.getInf()) / 20;
```

```
final AdvancedTimeScale timeScale = new AdvancedTimeScale(
      tmpds.getInf() - border, tmpds.getSup() + border,
      display.getWidth() - 1);
AdvancedTimeScale overviewTimeScale = new AdvancedTimeScale(timeScale);
RangeAdapter rangeAdapter = new RangeAdapter(overviewTimeScale,
      timeScale);

timeScale.setAdjustDateRangeOnResize(true);
timeScale.addChangeListener(new ChangeListener() {
   public void stateChanged(ChangeEvent e) {
      vis.run(DemoEnvironmentFactory.ACTION_UPDATE);
   }
});

// --------------------------------------------------------------------
// STEP 2: set up renderers for the visual data
ShapeRenderer dotRenderer = new ShapeRenderer(8);
DefaultRendererFactory rf = new DefaultRendererFactory(dotRenderer);
rf.add(new InGroupPredicate(GROUP_AXIS_LABELS), new AxisRenderer(
      Constants.FAR_LEFT, Constants.CENTER));
vis.setRendererFactory(rf);

// --------------------------------------------------------------------
// STEP 3: create actions to process the visual data
TimeAxisLayout time_axis = new TimeAxisLayout(GROUP_DATA, timeScale);

AxisLayout y_axis = new AxisLayout(GROUP_DATA, COL_DATA,
    Constants.Y_AXIS,
      VisiblePredicate.TRUE);

// add value axis labels and horizontal grid lines
AxisLabelLayout y_labels = new AxisLabelLayout(GROUP_AXIS_LABELS,
    y_axis);

// color must be set -> otherwise nothing displayed
ColorAction color = new ColorAction(GROUP_DATA, VisualItem.FILLCOLOR,
      ColorLib.rgb(100, 100, 255));

ShapeAction shape = new ShapeAction(GROUP_DATA,
    Constants.SHAPE_ELLIPSE);


// runs on layout updates (e.g., window resize, pan)
ActionList update = new ActionList();
```

```
        update.add(time_axis);
        update.add(y_axis);
        update.add(y_labels);
        update.add(new RepaintAction());
        vis.putAction(DemoEnvironmentFactory.ACTION_UPDATE, update);

        // runs once (at startup)
        ActionList draw = new ActionList();
        draw.add(update);
        draw.add(color);
        draw.add(shape);
        draw.add(new RepaintAction());
        vis.putAction(DemoEnvironmentFactory.ACTION_INIT, draw);

        // --------------------------------------------------------------------
        // STEP 4: set up a display and controls

        // ensure there is space on left for tick mark label (FAR_LEFT setting)
        display.setBorder(BorderFactory.createEmptyBorder(7, 25, 7, 0));

        // ensure (horizontal) grid lines are in back of data items
        display.setItemSorter(new ItemSorter() {
           public int score(VisualItem item) {
              int score = super.score(item);
              if (item.isInGroup(GROUP_AXIS_LABELS))
                 score--;
              return score;
           }
        });

        // show value in tooltip
        display.addControlListener(new ToolTipControl(COL_DATA));

        // --------------------------------------------------------------------
        // STEP 5: launching the visualization

        DemoEnvironmentFactory env = new DemoEnvironmentFactory("dot plot");
        env.setPaintWeekends(false);
        env.show(display, rangeAdapter);
    }
}
```

List of Figures

List of Tables

List of Listings

Bibliography

Aigner, W. [2006]. *Visualization of Time and Time-Oriented Information: Challenges and Conceptual Design*. Ph.D. thesis, TU Wien, Supervisors S. Miksch and H. Schumann.

Aigner, W., Federico, P., Gschwandtner, T., Miksch, S., and Rind, A. [2012]. Challenges of time-oriented visual analytics in healthcare. In Caban, J. J. and Gotz, D., editors, *Proc. VisWeek 2012 Workshop on Visual Analytics in Healthcare*, pages 17–20.

Aigner, W., Kainz, C., Ma, R., and Miksch, S. [2011a]. Bertin was right: An empirical evaluation of indexing to compare multivariate time-series data using line plots. *Computer Graphics Forum*, 30(1):215–228, doi: 10.1111/j.1467-8659.2010.01845.x.

Aigner, W. and Miksch, S. [2006]. CareVis: Integrated visualization of computerized protocols and temporal patient data. *Artificial Intelligence in Medicine*, 37(3):203–218, doi: 10.1016/j.artmed.2006.04.002.

Aigner, W., Miksch, S., Schumann, H., and Tominski, C. [2011b]. *Visualization of Time-Oriented Data*. Springer, London, doi: 10.1007/978-0-85729-079-3.

Aigner, W., Miksch, S., Thurnher, B., and Biffl, S. [2005]. PlanningLines: Novel glyphs for representing temporal uncertainties and their evaluation. In *Proc. Int. Conf. Information Visualisation (IV)*, pages 457–463. IEEE, doi: 10.1109/IV.2005.97.

Allen, J. F. [1983]. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, doi: 10.1145/182.358434.

Amor-Amorós, A., Federico, P., Miksch, S., Zambanini, S., Brenner, S., and Sablatnig, R. [2017]. Visual analytics for multitemporal aerial image georeferencing. In Sedlmair, M. and Tominski, C., editors, *Proc. 8th Int. EuroVis Workshop on Visual Analytics (EuroVA)*, pages 55–59. Eurographics, doi: 10.2312/eurova.20171120.

Auber, D. [2004]. Tulip – a huge graph visualization framework. In Jünger, M. and Mutzel, P., editors, *Graph Drawing Software*, pages 105–126. Springer, Berlin, doi: 10.1007/978-3-642-18638-7_5.

Auber, D., Archambault, D., Bourqui, R., Lambert, A., Mathiaut, M., Mary, P., Delest, M., Dubois, J., and Mélançon, G. [2012]. The tulip 3 framework: A scalable software library for information visualization applications based on relational data. Research Report RR-7860, INRIA. http://hal.inria.fr/hal-00659880 (last accessed Nov 2, 2017).

Bastian, M., Heymann, S., and Jacomy, M. [2009]. Gephi: An open source software for exploring and manipulating networks. In *Proc. Int. AAAI Conf. Weblogs and Social Media*. http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154 (last accessed Oct 25, 2017).

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

Bederson, B. B., Grosjean, J., and Meyer, J. [2004]. Toolkit design for interactive structured graphics. *IEEE Trans. Software Engineering*, 30(8):535–546, doi: 10.1109/TSE.2004.44.

Bettini, C., Jajodia, S., and Wang, S. X. [2000]. *Time Granularities in Databases, Data Mining, and Temporal Reasoning*. Springer, Berlin, doi: 10.1007/978-3-662-04228-1.

Bögl, M. [2013]. Visual Analytics for Time Series Analysis. Diplomarbeit, TU Wien, Supervisors P. Filzmoser and S. Miksch.

Bögl, M., Aigner, W., Filzmoser, P., Gschwandtner, T., Lammarsch, T., Miksch, S., and Rind, A. [2015]. Integrating predictions in time series model selection. In Bertini, E. and Roberts, J. C., editors, *Proceedings of the EuroVis Workshop on Visual Analytic, EuroVA*, pages 73–77. doi: 10.2312/eurova.20151107.

Bögl, M., Aigner, W., Filzmoser, P., Lammarsch, T., Miksch, S., and Rind, A. [2013]. Visual analytics for model selection in time series analysis. *IEEE Trans. Visualization and Computer Graphics*, 19(12):2237–2246, doi: 10.1109/TVCG.2013.222.

Bostock, M. and Heer, J. [2009]. Protovis: A graphical toolkit for visualization. *IEEE Trans. Visualization and Computer Graphics*, 15(6):1121–1128, doi: 10.1109/TVCG.2009.174.

Bostock, M., Ogievetsky, V., and Heer, J. [2011]. D3: Data-driven documents. *IEEE Trans. Visualization and Computer Graphics*, 17(12):2301–2309, doi: 10.1109/TVCG.2011.185.

Brandes, U., Eiglsperger, M., and Lerner, J. [n.d.]. GraphML Primer. http://graphml.graphdrawing.org/primer/graphml-primer.html (last accessed Nov 2, 2017).

Card, S. K., Mackinlay, J. D., and Shneiderman, B., editors [1999]. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, San Francisco.

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

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

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

Federico, P., Hoffmann, S., Rind, A., Aigner, W., and Miksch, S. [2014]. Qualizon Graphs: Space-efficient time-series visualization with qualitative abstractions. In *Proc. 2014 Int. Working Conf. Advanced Visual Interfaces, AVI*, pages 273–280. ACM, doi: 10.1145/2598153.2598172.

Fekete, J.-D. [2004]. The InfoVis toolkit. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pages 167–174. doi: 10.1109/INFVIS.2004.64.

Fekete, J.-D. [2013]. Visual analytics infrastructures: From data management to exploration. *Computer*, 46(7):22–29, doi: 10.1109/MC.2013.120.

Fekete, J.-D., Hemery, P.-L., Baudel, T., and Wood, J. [2011]. 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. doi: 10.1109/VAST.2011.6102446.

Fernández-Prieto, D., Naranjo-Valero, C., Hernández, J. T., and Hagen, H. [2017]. STRAD Wheel: Web-based library for visualizing temporal data. *IEEE Computer Graphics and Applications*, 37(2):99–105, doi: 10.1109/MCG.2017.27.

Fialli, J. and Vajjhala, S. [2006]. JSR 222: JavaTM Architecture for XML Binding (JAXB) 2.0. Java Specification Requests. https://jcp.org/en/jsr/detail?id=222 (last accessed Nov 7, 2017).

Forbes, A. G., Höllerer, T., and Legrady, G. [2010]. behaviorism: A framework for dynamic data visualization. *IEEE Trans. Visualization and Computer Graphics*, 16(6):1164–1171, doi: 10.1109/TVCG.2010.126.

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

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

Goralwalla, I. A., Leontiev, Y., Özsu, M., Szafron, D., and Combi, C. [2001]. Temporal granularity: Completing the puzzle. *Journal of Intelligent Information Systems*, 16(1):41–63, doi: 10.1023/A:1008788926897.

Goralwalla, I. A., Özsu, M. T., and Szafron, D. [1998]. An object-oriented framework for temporal data models. In Etzion, O., Jajodia, S., and Sripada, S., editors, *Temporal Databases: Research and Practice*, LNCS 1399, pages 1–35. Springer, Berlin, doi: 10.1007/BFb0053696.

Havre, S., Hetzler, E., Whitney, P., and Nowell, L. [2002]. ThemeRiver: Visualizing thematic changes in large document collections. *IEEE Trans. Visualization and Computer Graphics*, 8(1):9–20, doi: 10.1109/2945.981848.

Heer, J. and Agrawala, M. [2006]. Software design patterns for information visualization. *IEEE Trans. Visualization and Computer Graphics*, 12(5):853–860, doi: 10.1109/TVCG.2006.178.

Heer, J. and Bostock, M. [2010]. Declarative language design for interactive visualization. *IEEE Trans. Visualization and Computer Graphics*, 16(6):1149–1156, doi: 10.1109/TVCG.2010.144.

Heer, J., Card, S. K., and Landay, J. A. [2005]. prefuse: A toolkit for interactive information visualization. In *Proc. SIGCHI Conf. Human Factors in Computing Systems (CHI)*, pages 421–430. ACM, doi: 10.1145/1054972.1055031.

Heer, J., Kong, N., and Agrawala, M. [2009]. Sizing the horizon: The effects of chart size and layering on the graphical perception of time series visualizations. In *Proc. ACM SIGCHI Conf. Human Factors in Computing Systems (CHI)*, pages 1303–1312. ACM, doi: 10.1145/1518701.1518897.

Huynh, D. F., Karger, D. R., and Miller, R. C. [2007]. Exhibit: Lightweight structured data publishing. In *Proc. 16th Int. Conf. World Wide Web (WWW)*, pages 737–746. ACM, doi: 10.1145/1242572.1242672.

Isenberg, T., Isenberg, P., Chen, J., Sedlmair, M., and Möller, T. [2013]. A systematic review on the practice of evaluating visualization. *IEEE Trans. Visualization and Computer Graphics*, 19(12):2818–2827, doi: 10.1109/TVCG.2013.126.

Janetzko, H., Sacha, D., Stein, M., Schreck, T., Keim, D. A., and Deussen, O. [2014]. Feature-driven visual analytics of soccer data. In *Proc. IEEE Conf. Visual Analytics Science and Technology (VAST)*, pages 13–22. doi: 10.1109/VAST.2014.7042477.

Javed, W., McDonnel, B., and Elmqvist, N. [2010]. Graphical perception of multiple time series. *IEEE Trans. Visualization and Computer Graphics*, 16(6):927–934, doi: 10.1109/TVCG.2010.162.

Jensen, C., Dyreson, C., Böhlen, M., Clifford, J., Elmasri, R., Gadia, S., Grandi, F., Hayes, P., Jajodia, S., Käfer, W., Kline, N., Lorentzos, N., Mitsopoulos, Y., Montanari, A., Nonen, D., Peressi, E., Pernici, B., Roddick, J., Sarda, N., Scalas, M., Segev, A., Snodgrass, R., Soo, M., Tansel, A., Tiberio, P., and Wiederhold, G. [1998]. The consensus glossary of temporal database concepts – February 1998 version. In Etzion, O., Jajodia, S., and Sripada, S., editors, *Temporal Databases: Research and Practice*, pages 367–405. Springer, Berlin, doi: 10.1007/BFb0053710.

Kandel, S., Paepcke, A., Hellerstein, J., and Heer, J. [2011]. Wrangler: interactive visual specification of data transformation scripts. In *Proc. ACM SIGCHI Conf. Human Factors in Computing Systems (CHI)*, pages 3363–3372. ACM, doi: 10.1145/1979442.1979444.

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

Kerren, A., Purchase, H. C., and Ward, M. O., editors [2014]. *Multivariate Network Visualization*. LNCS 8380. Springer, Cham, doi: 10.1007/978-3-319-06793-3.

Kosara, R. and Miksch, S. [2001]. Metaphors of movement: A visualization and user interface for time-oriented, skeletal plans. *Artificial Intelligence in Medicine*, 22(2):111–131, doi: 10.1016/S0933-3657(00)00103-2.

Kuhail, M. A. and Lauesen, S. [2012]. 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.

Kuhail, M. A., Pandazo, K., and Lauesen, S. [2012]. Customizable time-oriented visualizations. In Bebis, G., Boyle, R., Parvin, B., Koracin, D., Fowlkes, C., Wang, S., Choi, M.-H., Mantler, S., Schulze, J., Acevedo, D., Mueller, K., and Papka, M., editors, *Advances in Visual Computing, Proc. ISVC 2012, Part II*, LNCS 7432, pages 668–677, Berlin. Springer, doi: 10.1007/978-3-642-33191-6_66.

Lammarsch, T. [2010]. *Facets of Time: Making the Most of Time's Structure in Interactive Visualization*. Ph.D. thesis, TU Wien, Supervisor S. Miksch.

Lammarsch, T., Aigner, W., Bertone, A., Gärtner, J., Mayr, E., Miksch, S., and Smuc, M. [2009]. Hierarchical temporal patterns and interactive aggregated views for pixel-based visualizations. In *Proc. 13th Int. Conf. Information Visualisation (IV)*, pages 44–50. IEEE, doi: 10.1109/IV.2009.52.

Lammarsch, T., Aigner, W., Bertone, A., Miksch, S., and Rind, A. [2011]. Towards a concept how the structure of time can support the visual analytics process. In Miksch, S. and Santucci, G., editors, *Proc. Int. Workshop Visual Analytics (EuroVA) in conjunction with EuroVis*, pages 9–12, Goslar, Germany. Eurographics, doi: 10.2312/PE/EuroVAST/EuroVA11/009-012.

Lammarsch, T., Aigner, W., Bertone, A., Miksch, S., and Rind, A. [2014]. Mind the time: Unleashing temporal aspects in pattern discovery. *Computers & Graphics*, 38:38–50, doi: 10.1016/j.cag.2013.10.007.

Lins, L., Klosowski, J. T., and Scheidegger, C. [2013]. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Trans. Visualization and Computer Graphics*, 19(12):2456–2465, doi: 10.1109/TVCG.2013.179.

Munzner, T. [2009]. A nested model for visualization design and validation. *IEEE Trans. Visualization and Computer Graphics*, 15(6):921–928, doi: 10.1109/TVCG.2009.111.

Munzner, T. [2014]. *Visualization Analysis and Design*. AK Peters/CRC, Boca Raton.

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

Perin, C., Vernier, F., and Fekete, J.-D. [2013]. Interactive Horizon Graphs: Improving the compact visualization of multiple time series. In Brewster, S., Bødker, S., Baudisch, P., and Beaudouin-Lafon, M., editors, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3217–3226, New York, NY, USA. ACM, doi: 10.1145/2470654.2466441.

R Core Team [2013]. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org/ (last accessed Nov 7, 2017).

Reas, C. and Fry., B. [2014]. *Processing: A Programming Handbook for Visual Designers*. MIT Press, second edition.

Reijner, H. [2008]. The development of the horizon graph. In Bartram, L., Stone, M., and Gromala, D., editors, *Proc. Vis08 Workshop From Theory to Practice: Design, Vision and Visualization*. http://www.stonesc.com/Vis08_Workshop/DVD/Reijner_submission.pdf (last accessed Nov 2, 2017).

Rind, A., Aigner, W., Miksch, S., Wiltner, S., Pohl, M., Turic, T., and Drexler, F. [2011]. Visual exploration of time-oriented patient data for chronic diseases: Design study and evaluation. In Holzinger, A. and Simonic, K., editors, *Information Quality in e-Health, Proc. USAB 2011*, LNCS 7058, pages 301–320, Heidelberg. Springer, doi: 10.1007/978-3-642-25364-5_22.

Rind, A., Federico, P., Gschwandtner, T., Aigner, W., Doppler, J., and Wagner, M. [2017]. Visual analytics of electronic health records with a focus on time. In Rinaldi, G., editor, *New Perspectives in Medical Records: Meeting the Needs of Patients and Practitioners*, TELe-Health, pages 65–77. Springer, Cham, doi: 10.1007/978-3-319-28661-7_5.

Rind, A., Lammarsch, T., Aigner, W., Alsallakh, B., and Miksch, S. [2013]. TimeBench: A data model and software library for visual analytics of time-oriented data. *IEEE Trans. Visualization and Computer Graphics*, 19(12):2247–2256, doi: 10.1109/TVCG.2013.206.

Saito, T., Miyamura, H. N., Yamamoto, M., Saito, H., Hoshiya, Y., and Kaseda, T. [2005]. Two-tone pseudo coloring: Compact visualization for one-dimensional data. In *Proc. 2005 IEEE Symp. Information Visualization (InfoVis 2005)*, pages 173–180. doi: 10.1109/INFVIS.2005.1532144.

Satyanarayan, A. and Heer, J. [2014]. Lyra: An interactive visualization design environment. *Computer Graphics Forum*, 33(3):351–360, doi: 10.1111/cgf.12391.

Satyanarayan, A., Moritz, D., Wongsuphasawat, K., and Heer, J. [2017]. Vega-Lite: A grammar of interactive graphics. *IEEE Trans. Visualization and Computer Graphics*, 23(1):341–350, doi: 10.1109/TVCG.2016.2599030.

Satyanarayan, A., Russell, R., Hoffswell, J., and Heer, J. [2016]. Reactive Vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE Trans. Visualization and Computer Graphics*, 22(1):659–668, doi: 10.1109/TVCG.2015.2467091.

Shannon, P., Markiel, A., Ozier, O., Baliga, N. S., Wang, J. T., Ramage, D., Amin, N., Schwikowski, B., and Ideker, T. [2003]. Cytoscape: A software environment for integrated models of biomolecular interaction networks. *Genome Research*, 13(11):2498–2504, doi: 10.1101/gr.1239303.

Shneiderman, B. [1996]. The eyes have it: A task by data type taxonomy for information visualizations. In *Proc. IEEE Symp. Visual Languages (VL)*, pages 336–343. doi: 10.1109/VL.1996.545307.

Shneiderman, B. and Plaisant, C. [2006]. Strategies for evaluating information visualization tools: Multi-dimensional in-depth long-term case studies. In *Proc. 2006 AVI*

Workshop BEyond time and errors: novel evaLuation methods for Information Visualization (BELIV '06), pages 38–43. ACM, doi: 10.1145/1168149.1168158.

Shumway, R. H. and Stoffer, D. S. [2011]. *Time Series Analysis and Its Applications: With R Examples*. Springer, 3rd edition.

Stolte, C., Tang, D., and Hanrahan, P. [2002]. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Trans. Visualization and Computer Graphics*, 8(1):52–65, doi: 10.1109/2945.981851.

The JUNG Development Team [2006]. JUNG: The Java Universal Network/Graph Framework. http://jung.sourceforge.net (last accessed Oct 29, 2017).

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

Urgun, B., Dyreson, C. E., Snodgrass, R. T., Miller, J. K., Kline, N., Soo, M. D., and Jensen, C. S. [2007]. Integrating multiple calendars using tauZaman. *Software: Practice and Experience*, 37(3):267–308, doi: 10.1002/spe.765.

Wagner, M., Rind, A., Thür, N., and Aigner, W. [2017]. A knowledge-assisted visual malware analysis system: Design, validation, and reflection of KAMAS. *Computers & Security*, 67:1–15, doi: 10.1016/j.cose.2017.02.003.

Weaver, C. [2004]. Building highly-coordinated visualizations in Improvise. In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pages 159–166. doi: 10.1109/INFVIS.2004.12.

Weaver, C., Fyfe, D., Robinson, A., Holdsworth, D., Peuquet, D., and MacEachren, A. M. [2006]. Visual analysis of historic hotel visitation patterns. In *2006 IEEE Symp. Visual Analytics Science And Technology (VAST)*, pages 35–42. doi: 10.1109/VAST.2006.261428.

Weber, M., Alexa, M., and Müller, W. [2001]. Visualizing time-series on spirals. In *Proc. IEEE Symp. Information Visualization, InfoVis*, pages 7–13. doi: 10.1109/INFVIS.2001.963273.

Weishapl, P. [2007]. TimeVis: Visualizing temporal data using prefuse. Bachelor thesis, TU Wien, Supervisor W. Aigner. http://www.cvast.tuwien.ac.at/sites/default/files/TimeVis.pdf (last accessed Nov 2, 2017).

Wongsuphasawat, K., Moritz, D., Anand, A., Mackinlay, J. D., Howe, B., and Heer, J. [2016]. Voyager: Exploratory analysis via faceted browsing of visualization

recommendations. *IEEE Trans. Visualization and Computer Graphics*, 22(1):649–658, doi: 10.1109/TVCG.2015.2467191.

Yalçin, M. A., Elmqvist, N., and Bederson, B. B. [2016]. Keshif: Out-of-the-box visual and interactive data exploration environment. In *IEEE VIS Workshop on Visualization in Practice*. http://adilyalcin.me/academic/KeshifInPractice.pdf (last accessed Feb 7, 2017).