Event-Based Visualization for User-Centered Visual Analysis

Dissertation zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)
der Fakultät für Informatik und Elektrotechnik der Universität Rostock

vorgelegt von
Dipl.-Inf. Christian Tominski
geboren am 14. Juli 1977 in Greifswald
wohnhaft in Rostock

Rostock, 09. Juni 2006

Principal Advisor

Prof. Dr.-Ing. habil. Heidrun Schumann, University of Rostock, Germany

External Reviewers

Univ.-Prof. Dr. Silvia Miksch, Danube University Krems, Austria
Dr. James Abello, DIMACS, Rutgers University, USA

Day of Defense

November 8, 2006

Abstract

In recent years, visualization has become an increasingly important tool to support exploration and analysis of larger volumes of data. Still, the interests of the users are only rarely considered.

Therefore, it is the goal of this thesis to shift the needs of users into the focus. For doing so, an event-based approach to visualization is pursued. This approach allows users to specify their interests as event types. During the visualization, instances of the specified event types are detected. It is then possible to automatically adjust visual representations according to the detected event instances. This process results in visualizations that are adapted to the needs and interests of users. Hence, users are supported in achieving their task at hand.

In this thesis, a generalized model of event-based visualization is developed. Furthermore, it is demonstrated how the developed methods and concepts can be implemented in an event-based visualization framework. Several event-enhanced visualizations are introduced to substantiate the claims made.

Zusammenfassung

In den vergangenen Jahren ist die Visualisierung zu einem wichtigen Hilfsmittel für die Exploration und Analyse größer Datenmengen geworden. Dennoch finden die Interessen der Anwender heute nur wenig Berücksichtigung.

Daher ist es das Ziel dieser Arbeit, die Anforderungen der Nutzer in den Fokus zu rücken. Hierzu wird ein ereignisorientierter Visualisierungsansatz verfolgt. Dieses Konzept erlaubt es den Nutzern, ihre Interessen mittels Ereignistypen zu spezifizieren. Instanzen dieser Ereignistypen können dann während der Visualisierung detektiert werden. Dadurch wird es möglich, beim Auftreten von Ereignisinstanzen die visuellen Repräsentationen automatisch so anzupassen, dass sie die Interessen des jeweiligen Anwenders widerspiegeln. Somit unterstützt die ereignisorientierte Visualisierung den Nutzer bei der Exploration und Analyse großer Datenmengen.

In der vorliegenden Arbeit wird ein allgemeines Konzept zur ereignisorientierten Visualisierung eingeführt. Darüber hinaus wird aufgezeigt, wie sich die entwickelten Methoden und Konzepte in einem ereignisorientierten Visualisierungsframework implementieren lassen. Um die Anwendbarkeit ereignisorientierter Visualisierungen zu untermauern, wird eine Reihe von Techniken vorgestellt, die durch Ereigniskonzepte erweitert wurden.

CR-Classification

H.5.0 Information Interfaces and Presentation – *General*, **I.3.6.** Computer Graphics – *Methodology and Techniques*, **I.3.8.** Computer Graphics – *Applications*

Keywords

Event-Based Visualization, User-Centered Visual Analysis, Multivariate Data, Spatio-Temporal Data, Graphs

Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my gratitude to all the people who helped me in conducting this thesis. A special thank you goes to my supervisor Heidi Schumann. She gave me the opportunity to work as a PhD student at her chair of Computer Graphics. Her guiding support and constructive criticism were very conducive to the success of this thesis. Special thanks also go to James Abello, who introduced me to new interesting areas of science. I am also grateful to him for inviting me for a two month research stay at DIMACS, Rutgers University. I also want to thank our friends in Vienna Silvia Miksch and Wolfgang Aigner for their hospitality and their support.

I am deeply grateful to my colleagues and friends. Petra, René, Georg, Thomas, Hermann, Mathias, Bernd, Peter, Hans-Jörg and Rosi – thank you for your support, inspiration, and advise, without you the work on this thesis would not have been half as exciting as it was. Thanks go to my students Clemens Holzhüter, Malte Willert, Falko Löffler, and Raphael Schmitz, who contributed valuable implementations. I would like to thank Ingo Wolf, Ciarán Close, Fabian Bünger, and Nikki Spencer for proofreading this thesis. Thanks go also to the colleagues I bothered with this task.

Lastly, and most importantly, I want to thank my parents and my beloved girlfriend Anja for their understanding, patience, and encouragement when it was most required. This thesis could not have been written without your support.

The work on this thesis was financially supported by the Landesgraduiertenförderung Mecklenburg-Vorpommern and the DFG Graduiertenkolleg 466.

Danksagung

An dieser Stelle möchte ich allen danken, die mir bei meiner Dissertation hilfreich zur Seite standen. Insbesondere danke ich Heidi Schumann. Sie eröffnete mir die Möglichkeit, als Promotionsstudent an ihrem Lehrstuhl zu arbeiten. Mit ihrer zielführenden Unterstützung und ihrer konstruktiven Kritik hat sie maßgeblich zum Gelingen dieser Arbeit beigetragen. Besonderer Dank gilt auch James Abello, der mir eine Tür zu weiteren interessanten Forschungsgebieten geöffnet hat. Ich danke ihm ebenfalls für die Einladung zu einem zweimonatigen Forschungsaufenthalt in den USA. Ich möchte auch unseren Wiener Freunden Silvia Miksch und Wolfgang Aigner für ihre Herzlichkeit und Unterstützung danken.

Es ist mir wichtig auch meinen lieben Kollegen und Wegbegleitern zu danken. Petra, René, Georg, Thomas, Hermann, Mathias, Bernd, Peter, Hans-Jörg und Rosi – habt Dank für eure Unterstützung; ohne euch hätte das Promovieren nur halb soviel Spaß gemacht. Dank gilt auch meinen Studenten Clemens Holzhüter, Malte Willert, Falko Löffler und Raphael Schmitz für ihre unterstützenden Arbeiten. Ich danke Ingo Wolf, Ciarán Close, Fabian Bünger, Nikki Spencer und allen Kollegen, die mir beim Korrekturlesen der Arbeit geholfen haben.

Es ist mir besonders wichtig, auch meinen Eltern und meiner geliebten Anja für ihr Verständnis, ihre Geduld, und ihre Unterstützung in allen Belangen zu danken. Ohne euch wäre diese Arbeit nicht möglich gewesen.

Diese Arbeit wurde finanziell gefördert durch die Landesgraduiertenförderung Mecklenburg–Vorpommern sowie das Graduiertenkolleg 466 der DFG.

1. Introduction

One of the main tasks of computers is handling of data. Database technology is a major research field of Computer Science that is driven by the requirement of flexibly managing ever increasing volumes of data. Nowadays, relational databases, which are the most frequently used technology for data management, are able to handle millions of data records, allow effective table-based data management, and enable fast access to the data. Therefore, relational database technology is being used by scientific institutions, public authorities, and companies of any size to collect a steadily increasing volume of data. Researchers from the University of California, Berkeley estimate that in 2002 about 5 Exabytes (= 5 Million Terabytes) of data were generated, of which 92% are available in digital form, mostly on hard disks [Lyman and Varian, 2003]. These data are potentially useful to their collectors: The data can help understanding physical phenomena, providing better health care, or planning new investments. To exploit this potential, it is necessary that the data are investigated.

However, studies have shown that only a fraction of the collected data is analyzed. This is due to the enormous efforts required to extract useful information from the data. Visualization is an aid in this process. Since human perception is highly dedicated to processing visual information, the aim is to reveal useful information by visually representing data. For doing so, data have to be mapped to visual primitives, which in turn can be rendered to computer-generated images. Provided that intuitive interaction techniques are used, visual representations can be explored and comprehended much easier than alphanumeric representations of data. Since the effort for data analysis is decreased significantly, interactive visualization is a widely accepted tool for drawing conclusions from data.

Problem statement

Although visualization has proved to be effective in various domains, there are still problems to be solved. The first challenge to mention is coping with constantly increasing volumes of data. When visualizing large datasets, the resulting visual representations are often overcrowded and visually cluttered. Even though this problem has been and is still being addressed in visualization research, most of today's visualization techniques reach their limits if a certain volume of data is exceeded. A further problem regards the applicability of visualization techniques. North et al. state that whereas relational database technology provides tools for flexibly managing collected data, such a high degree of flexibility has not yet been achieved in visualization [North et al., 2002]. Quite the contrary, visualization techniques are often tailored to only one particular visualization task that regards one particular dataset. This makes the available techniques rather inflexible. More general visualization systems that are ca-

pable of working on relational data structures are a more desirable solution [North et al., 2002], [Tang et al., 2004]. A third problem that must be addressed is the lack of relevance in visual representations. Amar and Stasko recently referred to this problem as *Worldview Gap*. Among other gaps, they describe the Worldview Gap as *...the gap between what is being shown in a visual representation and what actually needs to be shown to intuitively draw representational conclusions* [Amar and Stasko, 2005]. This statement gains special importance when considering users who are interested in different aspects of a dataset. In such cases, it is mandatory to take the different interests of users into account: The visual representations should be adjusted (by appropriate methods) to meet the users' specific needs.

The research presented in this thesis addresses the aforementioned problems, which can be summarized as decrease of effectiveness (due to overcrowded and cluttered visual representations), lack of flexibility (due to the limited applicability of visualization techniques), and lack of relevance in visual representations (due to the disregard of user interests). The goal of this thesis is to alleviate the identified problems, that is, to develop concepts that can help to **increase effectiveness, flexibility, and relevance** of visual representations.

Approach outline

The identified problems are tackled by involving users (i.e., their interests) in the visualization process. For doing so, an event-based approach to visualization is pursued. Event-based approaches have proved their usefulness in various application fields like software development, databases, or modeling and simulation. In these fields, events are commonly considered as occurrences of interest according to which automatic reactions have to be conducted. Event-related concepts have also been used in the context of visualization. Even though several event-related visualization approaches are known in literature, most of them are specific to a particular visualization problem, and hence cannot be applied in a general manner.

Speaking in terms of event-based visualization as introduced in this thesis, the basic idea is to let users specify their interest by means of event types, to detect instances of these events in the data, and to create visual representations automatically adjusted with respect to the detected event instances. Accordingly, three main aspects are investigated in this thesis in detail:

1. Event specification,
2. Event detection, and
3. Event representation.

To bridge the gap between informal user interests and the digital language of computers, a formalism for the event specification must be developed. Here, the difficulty is to build a formal basis that provides a suitable expressiveness while still allowing users to specify their interests as easily as possible. Especially when facing users who are not familiar with event-based visualization, it is necessary to provide methods and tools

that allow an intuitive specification of event types. Therefore, it is suggested to adapt formulas known from predicate logic and to build on them a model for user-centered event specification.

Once users have specified their interests as event types, concrete event instances may be detected from the data. The event detection methods to be described consider different types of events as well as specific needs of detecting events from static or dynamic datasets. To enable the efficient detection of event instances even in larger datasets, concepts for increasing the detection efficiency are particularly of interest.

The third aspect – the event representation – has major impact on achieving an increase in the effectiveness of visualizations. To allow an easy interpretation of event occurrences, any event representation should strive to highlight events (among the rest of the data) and, additionally, to make events of different types visually distinguishable. This thesis suggests two different concepts for representing event instances – implicit event representation and explicit event representation. Implicit event representations aim at automatically adjusting given visualization techniques. The major challenge in representing events implicitly is to adequately alter visualization parameters such that the stated goals are achieved. On the other hand, explicit event representations focus on visualizing only event occurrences, rather that the underlying data. Even though detaching events and data implies a high degree of abstraction, explicit event representation is an interesting alternative for gaining insight into even large datasets.

To make sense of the previously described aspects, it is necessary to compile the event specification, the event detection, and the event representation into a model of event-based visualization. Such a generalized model is the main contribution of this thesis. That model enables the application of user-centered visual representations in a wide range of scenarios, which is not the case for current event-related visualization approaches. To substantiate the generalized model of event-based visualization, a framework is introduced that implements the suggested model and provides several event-based visualization techniques. These techniques are capable of automatically reacting to events of interest to create visual representations that are adapted according to the users' tasks at hand. Furthermore, it is demonstrated how the concept of event-based visualization can be successfully applied beyond the borders of the suggested framework. Since the term *event* is often strongly related to time it is also important to mention that the approach developed in this thesis is not limited to visualizing only time-dependent data; in fact, events are used in a more general manner.

The usefulness of the concepts, techniques, and ideas summarized in this thesis is documented by several of international publications, of which the most relevant ones are [Tominski et al., 2004], [Tominski and Schumann, 2004], [Tominski et al., 2005b], and [Tominski et al., 2006a].

Thesis structure

The remainder of this thesis is structured as follows. In Chapter 2, a common basis for understanding the later chapters is built. This basis considers aspects of visualization as well as data-related and event-related aspects. Moreover, known visualization approaches that are related to events are reviewed. Based on that review, a classification

schema for event-related visualizations is introduced in Chapter 3. Chapter 4 starts with a description of unsolved problems and challenges of visualization research and states goals to be achieved in this work. Yet, the main concern of Chapter 4 is the development of the novel generalized model of event-based visualization. This model describes the integration of event methodology into the process of mapping data to visual representations. Based on a formal description of the model, methods for event specification and event detection are described in Sections 4.4 and 4.5. How events can be represented visually is examined in Section 4.6. An implementation of the proposed model is introduced in Chapter 5, where the event-based visualization framework *eVis* is described. In the same chapter it is also illustrated how the integration of events can increase the effectiveness of visualizations. This is done by the example of an event-enhanced TableLens and axes-based visualizations generated by the developed toolkit *VisAxes*. Moreover, it is explained how space-time paths can be utilized to facilitate high-level visualization tasks in the map-based visualization toolkit *TimeMap*. In Chapter 6 it is demonstrated that event-based visualization concepts can be successfully incorporated into further application scenarios, particularly in the context of graph visualization. Chapter 7 summarizes this thesis, states problems that remain unsolved, and gives an outlook on future work.

2. Basics and Related Work

This chapter describes the basics of visualization. It is shown how visualization is generally used to map data to visual representations. Moreover, introductions to relational databases and to general event-related notions are given. The last section of this chapter is dedicated to an analysis of current visualization approaches that are related to events.

2.1. Visually Communicating Information

> *Graphic aids for thinking have an ancient and venerable history.*
>
> [Card et al., 1999]

Revealing useful information in (abstract) data is not a trivial task. Especially when datasets are large and complex their analysis is challenging. In such cases, simple alphanumeric representations of the data can only be used to analyze few data items. Global relations within a dataset or even correlations between multiple data items are hard to detect from alphanumeric representations, since combining large amounts of numbers and words is very demanding for the human brain. On the contrary, the human visual system has enormous capabilities of perceiving and processing visual information. In fact, humans perceive their environment mainly by seeing. Amazingly, there appears to be only little thinking required to process the perceived information. This suggests the use of visualization for analyzing large and complex datasets.

The basic idea of visualization is to utilize the human visual system for facilitating the analysis of data [Card et al., 1999], [Ware, 2000], [Spence, 2001]. For doing so, data have to be mapped to visual representations. By using visual representations in combination with means of statistics [Moore and McCabe, 1999] or data mining [Ye, 2003], information contained in a dataset can be easily comprehended and even hidden information can be revealed.

To facilitate data analysis, the mapping from data to visual representations has to be done properly. By *properly* it is meant that a visualization must satisfy the three criteria [Schumann and Müller, 2000]:

- Expressiveness,
- Effectiveness, and
- Appropriateness.

A visualization is considered to be expressive if the relevant information of a dataset (and only this[1]) is expressed by the visualization. The term relevant implies that expressiveness of a visualization can only be assessed regarding a particular user working with the visual representation to achieve certain goals. A visualization is effective if it addresses the capabilities of the human visual system. Since perception, and hence the mental image of a visual representation, varies among users, effectiveness is user-dependent as well. Nonetheless, some general rules for effective visualization have been established in the visualization community. Appropriateness regards the tradeoff between efforts required for creating the visual representation and the benefits yielded by it. If this tradeoff is balanced, the visualization is considered to be appropriate.

Although expressiveness, effectiveness, and appropriateness are established criteria for visualization, an evaluation of visualization methods with regard to them is hard to perform. This is due to the fact that expressiveness and effectiveness are subjective and user-dependent [Tory and Möller, 2004]. Hence, empirical evaluation methods must be used to create significant results. Moreover, much effort is required to evaluate visualization methods with respect to appropriateness. Although the runtime of the applied algorithms and the required memory resources can be measured, the benefit of a visualization is hard to figure out. A large number of visualization tasks (see e.g., [Zhou and Feiner, 1998] or [Andrienko and Andrienko, 2006]) must be considered in this context. All these difficulties make the evaluation of visualization techniques a challenge for current and future visualization research.

The Visualization Transformation

This section will focus on how visual representations are generated in general. The process of mapping data to visual representations is called the visualization transformation. It can be defined as follows.

Definition 2.1 (Visualization transformation) *A mapping* $\texttt{vis} : \mathcal{D} \rightarrow \mathcal{V}$ *from a set* $\mathcal{D}$ *of datasets to a set* $\mathcal{V}$ *of visual representations, irrespective of whether these are static (i.e., images) or dynamic (e.g., animations or slide shows), is called a visualization transformation.*

Modeled like this, the visualization transformation is rather inflexible and applicable to only a limited number of visualization problems. Hence, *vis* is extended to allow a parameterization of the visualization transformations (see [Jankun-Kelly et al., 2002], [Jankun-Kelly, 2003]).

Definition 2.2 (Parameterized visualization transformation) *The extension of* $\texttt{vis}$ *to* $\texttt{pvis} : \mathcal{D} \times P_1 \times \cdots \times P_n \rightarrow \mathcal{V}$ *is called a parameterized visualization transformation. Here* $P_i \mid 0 \leq i \leq n, n \in \mathbb{N}$ *denote visualization parameter types (or visualization parameters for short), each of which consists of visualization parameter values.*

[1] It is crucial to avoid visualizations from which wrong conclusions could be drawn, i.e., that information could be perceived that is not contained in the data.

Technically, the visualization transformation is performed in four main steps – data analysis, filtering, mapping, and rendering (see Figure 2.1). They make up the visualization pipeline[2] [dos Santos and Brodlie, 2004]. The practical importance of the visualization pipeline is underlined by Chi's taxonomy of visualization techniques [Chi, 2000], which considers how different techniques implement the stages of the visualization pipeline.

To create a visual representation, a dataset is processed through the visualization pipeline as follows. In the data analysis step, the data are prepared for visualization (e.g., by applying a smoothing filter, interpolating missing values, or correcting erroneous measurements). In [dos Santos and Brodlie, 2004] this step is described as computer-centered, since it involves only little to no interaction with users. The filtering step performs a selection of data portions to be visualized. The selected portions are denoted as focus data. Usually, this step is user-centered. In the mapping step, focus data are mapped to geometric primitives (e.g., points, lines) and their attributes (e.g., color, position, size). The mapping step is the most critical one for achieving expressiveness and effectiveness, and is, therefore, the most interesting one to visualization designers. In the last step, geometric data are transformed to image data. Here, adequate rendering techniques are used to create the final visual representations (e.g., images or animations). The structure of the visualization pipeline allows a classification of the parameters $P_1, \ldots, P_n$ of the visualization transformation regarding the step they control. The fact that different visualization parameters affect different stages of the visualization pipeline will be considered in Section 4.6.

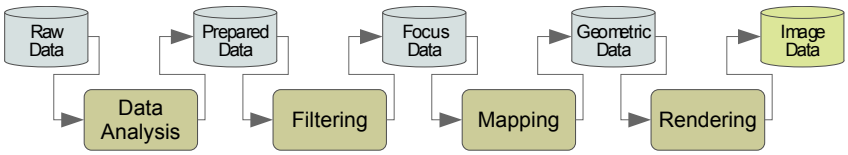


Figure 2.1.: The visualization pipeline [dos Santos and Brodlie, 2004].
The visualization pipeline describes the process of creating visual representation of data.

The Visualization Session

In the previous section it has been described how visualizations are created. This section details on how visualizations are used. A coarse categorization of application scenarios for visualization is given by:

Visual exploration The aim pursued with visual exploration is to give an overview of the data and to allow users to interactively browse through different portions of the data. In this scenario, users have no or only vague hypotheses about the data; their aim is to find some. In this sense, visual exploration can be understood as an undirected search for relevant information within the data. To support users

[2] This pipeline is an extension of the widely accepted reference model by Haber and McNabb [Haber and McNabb, 1990].

in the search process, a high degree of interactivity must be a key feature of visual exploration techniques.

Visual analysis The visual analysis aims at visually supporting the verification or falsification of hypotheses concerning a dataset. This means that users perform a directed search for information. A high degree of interactivity is required here as well. Since hypotheses are given a priori, visual analysis techniques should be able to guide the users during the search process. Visual analysis techniques are often tailored to one certain analysis problem (i.e., a limited set of hypotheses).

Visual presentation Visual presentation aims at communicating and sharing insights that have been gained via visual methods (e.g., to discuss the findings with other data analysis experts). The visual presentation must emphasize the relevant information among the rest of the data. This is of particular importance, because presentations are often realized on print media, which are very limited in terms of interaction.

The interactive use of visual representations is considered a visualization session [Jankun-Kelly et al., 2002]. By interactive it is meant that users control the appearance of the visual representations by changing the parameters provided by the visualization technique. Figure 2.2 depicts how users participate in a visualization session, which can be modeled as a system with feedback.

Such an integration of the user is advantageous for users familiar with visualization. They are able to control the visualization transformation. On the other hand, inexperienced users might have difficulties in understanding the way certain parameters influence the generation of the visual representation. Therefore, means for supporting users in setting the parameters and methods for automatically adjusting parameters are required. The event-based visualization model presented in this thesis is one possibility to achieve this requirement.

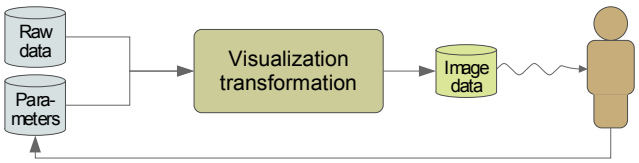


Figure 2.2.: The visualization session.
The figure depicts how the process of creating visual representations of data is embedded into an interactive visualization session. It is shown that users can influence the appearance of the visual representation by adjusting visualization parameters.

Concepts for Visualizing Large Datasets

In times of ever increasing volumes of data, designing sophisticated visualization techniques is a challenging task. When visualizing large datasets on computer displays two

competing goals must be achieved. Firstly, it is important to convey a general overview of all data. Secondly, it should be possible to analyze certain portions of the data in detail. Research in information visualization has resulted in commonly accepted concepts for improving the readability of information graphics and the way users access represented information. These concepts also address the challenge of visualizing large volumes of data. The basic ideas behind these concepts are:

Overview+detail One maxim in visualization is known as the visual information seeking mantra: *Overview first, zoom and filter, then details on demand* [Shneiderman, 1996]. This mantra suggests providing users with a coarse overview of the data and allowing a detailed analysis in separate views that are opened upon user request. Although this concept is capable of providing overview and detailed view, it requires additional cognitive effort to combine the views mentally. Furthermore, the views often overlap each other, which leads to occlusion of information.

Focus+context Combining overview and detailed view in one single visual representation is the aim of this concept. Here, the portion of the data being of interest to a user is considered the focus, and is represented in detail. The rest of the data is considered the context and is only coarsely represented. Even though this concept can be implemented in various ways [Björk et al., 1999], the most prominent ones are distortion-based techniques [Keahey, 1998]. Other approaches consider blurring techniques as well [Kosara, 2001]. Depending on the specific realization, the focus+context concept reduces the cognitive efforts to comprehend the visualized data and, furthermore, reduces occlusion of represented information.

Overview+detail and focus+context heavily rely on interaction performed by users. This underlines the need for highly interactive visualizations methods and is evidence for the necessity of involving the user in the visualization process.

The visualization-related terms, models, and concepts that have been described here are the basis for the development of a general event-based model of visualization. In later chapters, these terms and models are extended to achieve the set goals.

2.2. Data and Visualization

> *Database management systems are at the heart of current information system technology.*
>
> [Paton, 1999]

Visualization aims at gaining insight by visually representing data. This implies that the data are fundamental for any visualization. For this reason, research on visualization techniques, concepts, or methodologies must be grounded on a description of the addressed kind of data. Several data are distinguished in the field of visualization. Depending on data type, scaling of the value range, data dimensionality, and data structure the following distinctions are commonly accepted [Schumann and Müller, 2000]:

- Data type: scalar data, vector data, tensor data
- Scaling of the value range: qualitative (nominal, ordinal) or quantitative (discrete, continuous) scaling
- Data dimensionality: univariate, multivariate data
- Data structure: unstructured, lists, tables, grids, graphs

There is no technique that is capable of visualizing all kinds of data. In fact, visualizing some kinds of data is such a challenging task that separate research fields focusing on only one kind of data (e.g., volume visualization, graph visualization) have been established.

Besides knowing the kind of data to be visualized, it is also important to consider the means used for storing and accessing the data. The concept of modeling datasets as relations[3] between attributes has become the most commonly used approach to handling arbitrary data. Even graph structures are often represented in a relational manner. Relational data structures were first introduced by Edgar F. Codd in 1970 [Codd, 1970]. Since then, research on relational database technology has resulted in the development of several relational database management systems (RDBMS). Nowadays, RDBMS are used in many fields of information technology.

From the visualization point of view, relational databases have also become increasingly important. However, many of today's visualization tools are only able to access (mostly proprietarily formatted) files. This implies not only a lack of flexibility, but can also become a bottleneck if large volumes of data have to be visualized. On the contrary, even very large data sources can be accessed easily by taking advantage of relational database technology. Furthermore, RDBMS provide functionality for performing data-related tasks like selection, aggregation, or sorting efficiently. Another advantage is that RDBMS servers are capable of processing multiple data queries originated from multiple users on remote hosts. This enables scenarios where several users analyze the same data regarding different aspects of interest. Since more and more data are stored by means of RDBMS, database and visualization technology have to converge or at least should be coupled more tightly (see [Derthick et al., 1997], [Humphrey, 2000], [Tang et al., 2004]). Approaches like VisDB [Keim and Kriegel, 1994] or the more recent Polaris [Stolte et al., 2002] give evidence of the advantages of such a tight coupling.

In this thesis, relational concepts are used to describe and to access the data being visualized. The definition of events will also be built on relational terminology. By doing so, the broad formal basis from relational database research can be used.

A brief introduction to the very fundamentals of relational data structures will be given to ease the understanding of the definition of events presented in Chapter 4. The following introduction is based on [Maier, 1983] and is adapted to the requirements of the work presented here. More detailed introductions to relational databases can be found in [Elmasri and Navathe, 2003], [Date, 2003], or [Heuer and Saake, 2000].

[3] Basically, relations can be understood as data tables.

In relational database theory, *attributes* are named measurements (e.g., temperature or cloud cover in a climate-related dataset). More abstractly, attributes describe what is being observed. The *value range* associated with an attribute contains all values that could possibly be observed. Whereas the value range of a temperature attribute would be a subset of the real numbers $\mathbb{R}$, the value range of a cloud cover attribute could be, for instance, a discrete set of categorical values $C = \{clear, cloudy, covered\}$. Attributes and value ranges are formally defined as follows.

Definition 2.3 (Attributes and value ranges) *Attributes A_i are defined as elements of a finite set $\mathcal{A} = \{A_i \mid 0 \leq i \leq n, n \in \mathbb{N}\}$. Furthermore, value ranges VR_i are defined as elements of a finite set $\mathcal{VR} = \{VR_i \mid 0 \leq i \leq m, m \in \mathbb{N}\}$. Each attribute is associated with a value range by means of a mapping $\texttt{range} : \mathcal{A} \rightarrow \mathcal{VR}$. The union $\mathcal{U} = \bigcup_{i=0}^{m} VR_i$ of all value ranges is called the universe.*

To be able to describe a collection of attributes being observed, *relation schemas* are introduced. A particular observation of the attributes of a relation schema is modeled as a *tuple*. Several concrete observations are considered to be a *relation* over the relation schema. To be more descriptive, a relation can be seen as a table in which the relation schema determines the columns, and tuples represent the rows of the table.

Definition 2.4 (Relation schemas and relations) *A subset $\mathcal{T} \subseteq \mathcal{A}$ is called a relation schema. A relation T on $\mathcal{T}$ is a finite set of mappings $T = \{\texttt{t} \mid \texttt{t} : \mathcal{T} \rightarrow \mathcal{U},\ \texttt{t}(A_i) \in \texttt{range}(A_i),\ A_i \in \mathcal{T}\}$. The mappings $\texttt{t}$ are called tuples of the relation.*

Continuing the previous example, a relation schema for climate data could be $\mathcal{T} = \{$ date, temperature, cloud cover$\}$. An example for a specific relation T on $\mathcal{T}$ could be $T = \{\texttt{t}_1, \texttt{t}_2, \texttt{t}_3\}$, where the tuples are defined as:

$$\texttt{t}_1(A_i) = \begin{cases} \text{'2004-11-15'} & | \quad A_i = \text{date} \\ 11 & | \quad A_i = \text{temperature} \\ \text{cloudy} & | \quad A_i = \text{cloud cover} \end{cases}$$

$$\texttt{t}_2(A_i) = \begin{cases} \text{'2004-11-16'} & | \quad A_i = \text{date} \\ 9 & | \quad A_i = \text{temperature} \\ \text{cloudy} & | \quad A_i = \text{cloud cover} \end{cases}$$

$$\texttt{t}_3(A_i) = \begin{cases} \text{'2004-11-17'} & | \quad A_i = \text{date} \\ 7 & | \quad A_i = \text{temperature} \\ \text{covered} & | \quad A_i = \text{cloud cover} \end{cases}$$

It must be mentioned that, even though tuples of a relation can be represented as rows of a table, the tuples of a relation are not ordered by definition. In fact, tuples are not even related to each other; they are merely elements of a relation. However, in many cases the semantics of the data contained in a relational dataset implies relations between tuples. Time series are a prominent example for data where an order is implied by a time attribute in the relation schema.

The basic relational notions presented here are picked up in Chapter 4 to define the meaning of events used in this thesis. Specifically, they will be relevant in Section 4.4 where it is described how events can be specified, and in Section 4.5 where it is described how event detection can be realized.

2.3. Events as the Basis for Reactive Behavior

> *..., we will not attempt to formally define what an "event" is.*
>
> [Cassandras and Lafortune, 1999]

An event is a ubiquitous element of life. The term *event* is stressed in nearly every field of science, and hence, a diversity of meanings of events is known. It is the aim of this section to review meanings of notions of events. For doing so, descriptions of the term event as well as related notions from the Computer Science literature are presented.

In common literature, a variety of definitions of the term event exists. Usually, an event is considered to be something that happens, especially something unusual or important. Yet, most literature avoids defining the meaning of events precisely without indicating the field of application in which they are used. The Brockhaus Encyclopedia [Brockhaus, 2003] actually contains three entries for the term event. One is related to physics, where an event is considered a physical process with negligible spatial and temporal extensions; a second is related to probability theory, where an event is considered a subset of the set of all possible outcomes of a probability experiment. The third definition describes events in Computer Science as state changes or actions resulting in certain consequences.

Although a general understanding of the term event is given by these common descriptions, each subfield of Computer Science has got its own associations of the term event. Some of these are reviewed in the following sections.

Events in Artificial Intelligence

Events are always integrated in a temporal domain. For ages, scientists have been thinking hard about the meaning and implications of time. Nowadays a vast number of notions of time-related phenomena exist. The notions that are relevant in artificial intelligence are presented in [Hajnicz, 1996]. Among others the important notions of *fact* and *event* are described as follows:

Facts Facts are primitives that describe the state of a world. They are strongly connected to everything being static (i.e., does not change by itself). Since facts often describe objects of a world, they are sometimes also denoted as properties. This becomes comprehensible when imagining the fact "The car is red".
Facts can be either true or false at certain points of time. This means that for a red car the previously stated fact holds. After the car has been painted blue, the fact obviously does not hold anymore.

Events Whereas facts describe what is static, events are used to denote dynamic elements of a world. When events occur, they bring mobility to the facts, i.e., events cause change to whether facts hold or not. For instance, the event "Paint the red cars blue" would change the validity of the fact "The car is red" to false and the validity of the fact "The car is blue" to true.
Events can be either instantaneous or time consuming. Theoretical works generally use instantaneous events for the sake of simplicity, whereas real-world events (e.g., the car painting event) are commonly considered to be time consuming.
Since events might occur multiple times, it is necessary to distinguish particular event occurrences. Therefore, it is common to differentiate event types, which describe the event, and actual event instances, which are connected to the temporal primitives at which they occurred.

Actions and *processes* are described as basic time notions as well. However, the notions action and process are used very differently by different researchers. Moreover, both notions are sometimes mixed in their meaning. According to [Hajnicz, 1996] an action is an event that is performed by a living agent (e.g., a human), and a process is an intermediate notion between facts and events. Processes can be considered gradual, continuous, often very slow events.

This description of facts and events gives a first, yet abstract indication of the meaning of events. For details and further references, the interested reader is referred to [Hajnicz, 1996].

Events in Dynamic Systems

Modeling and simulation research, specifically research on discrete event systems (DES), deals with events and their implications on dynamic systems. Hence, research on DES can be considered fundamental for Computer Science. Usually, DES textbooks avoid formally defining what events are. Events are introduced more or less informally: In [Cassandras and Lafortune, 1999] it is emphasized that in DES *...an event should be thought of as occurring instantaneously and causing transitions from one state value to another*; in [Banks et al., 2001] major event-related notions *...are briefly defined and then illustrated by examples*. First, it is defined that the state of a system is a collection of variable values that fully describes the system at any time. An event is then considered an instantaneous occurrence that changes the state of a system. As such, events may be identified with a specific action taken (e.g., pressing a button), events may be viewed as a spontaneous occurrence dictated by nature, or an event may be the result of several conditions that are suddenly all met (e.g., all thresholds of some sensory devices are exceeded).

In DES, events are combined with a variety of theoretical concepts of Computer Science to model, analyze, control, evaluate, and/or optimize dynamic systems. What is astonishing about this is the fact that even though it is difficult to formally describe what events are, they are widely used in formal models like, for instance, finite automata, Petri nets, or Markov chains.

Commonly, events are modeled as elements of a discrete set $E = \{e_1, e_2, \ldots\}$ of all

possible events occurring in a dynamic system. To differentiate between an event type and a particular event instance, an actual event instance is usually identified by a pair (e_i, t_j) where $e_i \in E$ denotes the event type, and $t_j \in T$ denotes the point in some temporal domain T at which the event occurred. In this context, combinations of events are also introduced to describe ordered collections of occurred events.

One important application of events is to use them for modeling controlled systems. In this thesis it will be described how occurrences of events are used to somehow "control" the outcome of the visualization pipeline.

Events in Databases

It has already been mentioned that modern relational database management systems (RDBMS) are capable of handling enormous volumes of data. However, handling large datasets turned out to be difficult if certain constraints implied by the semantics of the data had to be enforced. A simple example is a bank account that must not decrease below zero. Procedures for enforcing this constraint must be implemented in every application that accesses the data.

Active database research aims at centralizing the monitoring of data constraints within the RDBMS (see [Paton, 1999], [Paton and Díaz, 1999]). The idea is to define active event-condition-action (ECA) rules. Such rules are evaluated by the RDBMS on occurrence of certain events and cause an action to be performed if some conditions hold. Each part of an ECA rule will be briefly described in the following:

Event The event part of an ECA rule describes on what occurrences the rule is considered to be of interest. Events can be for example the insertion, alteration, or deletion of data, but also the access to data or some external events. Although commonly only these primitive events are used, there is ongoing research to enable composite events as well [Gehani et al., 1992a], [Galton and Augusto, 2002].

Condition The condition defined for an ECA rule describes the context in which the rule is considered to be relevant. Conditions are usually specified in a descriptive manner (e.g., some SQL dialect). It is possible to omit the condition in cases where no context description is necessary.

Action The action described in an ECA rule is performed on occurrence of the event and positive evaluation of the condition. Since an action may result in changes to the data or even in a rejection of certain database operations, it could happen that the execution of an action causes further ECA rules to be triggered. Therefore, it is important to ensure that for a given set of active rules, all rules terminate in finite time for all possible relations of a database [Baralis, 1999].

Irrespective of a particular syntax for ECA rules, a rule to ensure a positive balance of a bank account might look like Figure 2.3. The application of active rules in RDBMS helps ensuring data consistency. The behavior implied by certain semantics can be defined directly in the RDBMS; it is no longer necessary to implement the behavior in particular applications redundantly.

```
ON change TO balance OF account
IF new.balance < 0
DO INSTEAD <cancel transaction>
```

Figure 2.3.: Example of an ECA rule.
The depicted ECA rule cancels any transaction that tries to change the balance of an account to below zero.

The research on active databases inspired the work presented in this thesis a lot. As it becomes clear in Chapter 4, using events in visualization raises similar challenges as stated for active databases.

The Essence of Events

In the previous sections, examples of how events are applied in Computer Science have been described. There are many more applications of events in Computer Science like, for instance, programming languages, software modeling, operating systems, and hardware design.

The different terms reviewed in the previous paragraphs can be categorized into notions that describe static characteristics (facts, state, condition) and dynamic aspects (event, action, process) of a considered world or system. What is common to most applications of events in Computer Science is that events are not used separately (i.e., detached from some system). Generally, events are embedded into an event-action schema, because it would not make sense to consider a happening as an event if it would neither be recognized nor reacted to. In conclusion, events are the basis for reactive behavior in computer systems. However, it is not always clear what is the cause and what is the effect in this schema. On the one hand, events are described as occurrences identified by some specific action taken (e.g., a button was pressed); on the other hand, events are the cause for specific actions to be taken (e.g., cancelation of a database transaction).

It is worth noting that event types and event instances are commonly distinguished. A differentiation between these terms is also important for the research presented in this thesis. The role of both terms can be briefly described as follows:

Event type An event type is an abstract description of conditions or circumstances under which an entity is considered to be of interest. In other words, event types describe why something is of interest. For instance, the event type "airplane take-off" can be described by the condition "altitude > 0".

Event instance Event instances (or short events) are actual occurrences of interest, usually described by a pair consisting of a specific event type and an entity of some observation space. Depending on the observation space, different event instances are possible. For the previously mentioned event type "airplane take-off", one can image an event instance in time ("airplane take-off", 6:37pm) or an event instance that considers flights ("airplane take-off", "Flight CO96").

The precise meaning of event types and event instance in the context of event-based visualization will be discussed in detail in Chapter 4.

2.4. Events in Visualization – State of the Art

> *..., the focus is automatically directed towards the special events.*
>
> [Reinders et al., 2001]

In recent years, a variety of visualization techniques have been developed (see [Card et al., 1999], [Ware, 2000], or [Spence, 2001]). However, the focus of most of these techniques is on visualizing data, rather than on the visualization of events that are somehow related to the data. This section is dedicated to reviewing the current state of the art of event-related visualization research. It can be said in advance that, similar to the things stated in the previous sections, the notion of events is considered in a variety of different meanings. To simplify the later assessment of the reviewed concepts and techniques, a classification schema for event-related visualizations is developed in Chapter 3.

2.4.1. Information Visualization and Events

This section will give a review of information visualization approaches that are connected to or make use of event-related terminology.

Feature Visualization and Events

Visualizing time-dependent flow data is a challenging task due to the vast volumes of data that must be processed. Reinders et al. have investigated this problem and published results of their research in a variety of substantial papers, which recently have been summarized in Reinders' thesis [Reinders, 2001]. A compact yet comprehensive overview of Reinder's thesis can be found in [Reinders et al., 2001].

Reinders et al. follow a feature-based approach combined with events to visualize time-dependent flow data. Their approach is divided into four basic steps – *feature extraction*, *feature tracking*, *event detection*, and *visualization*. In the following, these steps will be described in detail.

Feature extraction The task of feature extraction is to derive features for each time-step contained in the raw flow data. Reinders et al. understand features as *...apparent coherent structures* within the data. Roughly speaking, features are portions of the data complying with certain conditions. Such features can be extracted by various methods. Reinders et al. use a selective approach [van Walsum and Post, 1994], which integrates the user in the initial determination of features, i.e., users describe certain characteristics that must hold for a feature. Once features have been extracted, attributes are calculated for each feature. For doing so, volume integrals are used. This enables the calculation of attributes like center point, volume, mass, and average values of 3D ellipsoids. The accuracy

of the attribute calculation procedure has been evaluated, and identified to be sufficient even for noisy data [Reinders et al., 1998].
The authors state that by feature extraction and attribute calculation a higher level of abstraction is achieved, the volume of the data to be visualized is reduced, and a basis for feature tracking and event detection is established.

Feature tracking So far, features are calculated separately for each time-step. It is not yet possible to draw conclusions about the evolution of features. The aim of the feature tracking step is to find paths of features [Reinders et al., 1999]. To achieve this aim, the frame-to-frame correspondence problem must be solved. Reinders et al. follow an attribute correspondence approach, which considers the 3D ellipsoid attribute set. This means that position, velocity, as well as orientation correspondence must be calculated. For doing so, a prediction of the attributes of a feature for the time-step $t + 1$ is extrapolated from the time-step t. Then, all extracted features of time-step $t + 1$ are examined regarding their correspondence to the extrapolated prediction by testing it against a weighted sum of position, velocity, and orientation correspondence. If multiple features of time-step $t + 1$ pass the correspondence test, the one with the best confidence index is considered to correspond to the feature of time-step t.
It has been verified by Reinders et al. that this procedure of tracking features by calculating predictions and attribute correspondences is stable, and thus is suitable for the visualization of features of time-dependent flow data.

Event detection Whereas the feature tracking step determines the evolution of features, the event detection step aims at detecting specific events in the evolution of features. Reinders et al. describe events as *...any development in the evolution of a feature that is significant.* This implies that events are considered to be dependent on the application domain, i.e., significant events have to be figured out for each addressed application. For features in time-dependent flow data, several basic events are suggested. These are *continuation*, *birth* and *death*, *entry* and *exit*, *split* and *merge*, as well as the special event *unresolved*. Besides these, dedicated topology events are proposed – *topo-loop* as well as *topo-junction*.
In fact, the *continuation*[4] event is simply the continuation of a feature path calculated in the feature tracking step. Hence, no additional detection algorithm is required for this event.
On the contrary, the other event types apparently require special algorithms to detect them. Death events are expected for features that are small in "size" (mass or volume), further decrease in size, and then disappear. They are detected by comparing the size of a feature at time-step t with its predicted size at $t+1$. Additionally, the predicted size must be small with respect to a certain tolerance. Since birth events can be considered the opposite of death events, birth events are detected by backward prediction (i.e., by predicting $t-1$ from t) and negation of the stated conditions.
For a detailed description of the detection of entry, exit, split, and merge, as well

[4]Reinders also refers to the continuation event as non-event.

as the topological events, interested readers are referred to [Reinders, 2001]. Here, only the former four events shall be briefly explained. Apparently, entry and exit events are expected when features enter or leave the boundaries of the observed system. In some cases (e.g., flow through a pipe), entry and exit events cannot occur due to impermeability of the system's boundaries. Merge events occur if several features approach each other and continue as one. In this case, the "size" (mass or volume) of the merged feature is equal to the combined "sizes" of the merging features. The split event is defined as the opposite of the merge event.
The event detection process lifts the visualization of flow data to yet a higher level of abstraction. By focusing on events, the volume of data to be visualized is further reduced while significant changes in the evolution of flow data can still be conveyed.

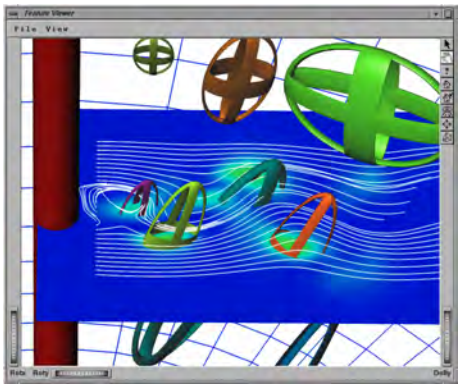


Figure 2.4.: Feature-based visualization of time-dependent flow data [Reinders, 2001]. The figure shows an iconic representation of flow features extracted from time-dependent flow data. By representing features as 3D icons, users are visually guided to the salient characteristics of the data. The visualization is enhanced by an additional streamline display.

Visualization Last but not least in the line of steps proposed by Reinders et al. is the visualization step. Basically, it focuses on visualizing features and events instead of visualizing the raw flow data. Two visualization techniques are used – a 3D iconic representation of features and a graph visualization of events (see Figure 2.4 and Figure 2.5, respectively).
An iconic representation of features (see [van Walsum et al., 1996]) can be computed by mapping attributes derived in the feature extraction step to *...3D amorphous objects of a size (much) smaller than the data set domain* [Reinders et al., 2001]. It is suggested to map the attributes of features to the shape of 3D ellipsoids, which have 9 degrees of freedom, and hence can provide a good indication of position, size, and orientation of features. Skeleton shapes are suggested for more complicated features [Reinders et al., 2000].
The constructed ellipsoid icons are then rendered in a 3D presentation space. By doing so, the characteristics of features are easily recognizable. The temporal

character of the data is reflected by animating the 3D ellipsoid icons. By this, the dynamic evolution of features can be conveyed. To support the exploration of the data, the animations can be controlled via a VCR-like user interface. The feature visualization can be enhanced by streamlines to further ease the understanding of the represented data.
Reinders et al. also suggest a graph-based representation of events. In this case, events are considered as vertices of a graph; edges of the graph are considered for events occurring subsequently at the same feature. To represent the constructed graph, Reinders et al. use a graph visualization algorithm that calculates a horizontal layout and minimizes edge-crossings in the visual representation. Since most of the subsequently occurred events are continuation events, they are represented as paths of features. Other events are visualized by small iconic indicators that are unique for each considered type of event. This representation of events abstracts from particular attributes of features, and thus focuses the attention of users on significant changes in the evolution of time-dependent flow features.

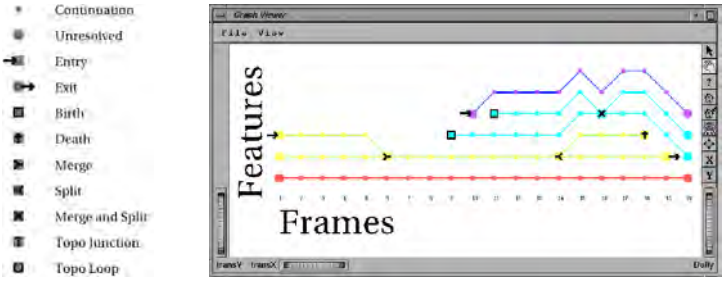


Figure 2.5.: Graph-based representation of events [Reinders, 2001].
Reinders et al. display events as nodes of a graph. The event type is conveyed by unique node icons. The graph as a whole represents the evolution of features in time.

With their four-step feature-based approach Reinders et al. have developed a powerful framework for visualizing vast volumes of time-dependent flow data. By successively extracting relevant information (from raw data to features, from features to events), different levels of abstraction are achieved. Especially the consideration of significant events allows a very simple, yet expressive visualization as a graph, which reveals the very characteristics in the analyzed flow. Unfortunately, the defined events are strongly connected to the features derived from the time-dependent flow data. An integration of the user into the specification of events has not been considered.

Events in Process Visualization

Matković et al. describe in [Matković et al., 2002a][5] a system that integrates information visualization technology and classic monitoring techniques. The authors state

[5] An extended version of the paper is available in [Matković et al., 2002b].

that, even though a variety of classic analog monitoring devices (e.g., gauge or bar instruments) have been implemented as virtual instruments for computer displays, the known systems do not benefit from the recent advances in information visualization. Therefore, several extensions to known virtual instruments are suggested. Furthermore, the integration of such instruments into an actual monitoring scenario is described.

A first extension considers increasing the amount of information represented by a virtual instrument. Since classic instruments are capable of showing only the current state of a monitored variable, Matković et al. enhance virtual instruments such that they allow the representation of a certain history of values. This eases the comprehension of the situation of the monitored system and is helpful when assessing critical states during a monitoring session.

To further benefit from information visualization technology, the focus+context concept is applied when representing a collection of virtual instruments. For doing so, virtual instruments that are capable of representing multiple monitored variables at different levels of detail have been designed. The suggested levels of detail for an instrument representing multiple data sources are:

1. Simple color-based indication of the average value
2. Textual representation of the average value
3. One single virtual instrument representing all monitored variables graphically
4. Several separate virtual instruments each dedicated to a single monitored data source

As depicted in Figure 2.6 the different representations occupy different amounts of screen space. This is exploited during the arrangement of the virtual instruments on the screen. Instruments monitoring variables of interest are represented at higher levels of detail (focus) to convey more information. Other instruments are shown at lower levels of detail (context).

Besides allowing users to interactively adjust the represented levels of detail (e.g., by using the mouse), the focus can also be altered automatically. This is where events come into the picture. Matković et al. consider predefined, undesired system states – more precisely the event that the system switches to one of these undesired states – for an implicit alteration of the focus. On occurrence of such events (e.g., a value exceeds a certain threshold) instruments representing the particular variable are put into the focus. By doing so, the focused instruments are not only filled up with additional information, but they are also made visually more present by their increase in size. Since the users' attention is attracted by bigger instruments representing undesired system states, necessary actions can be initiated more quickly.

However, the event specification is rather limited. Neither are events described that relate to more than one variable nor are combinations of events considered. Moreover, it is not described by the authors how users can actually specify their interests as events. Nonetheless, the approach presented in [Matković et al., 2002a] is a salient example of how events can be successfully applied to improve visualization. In this particular

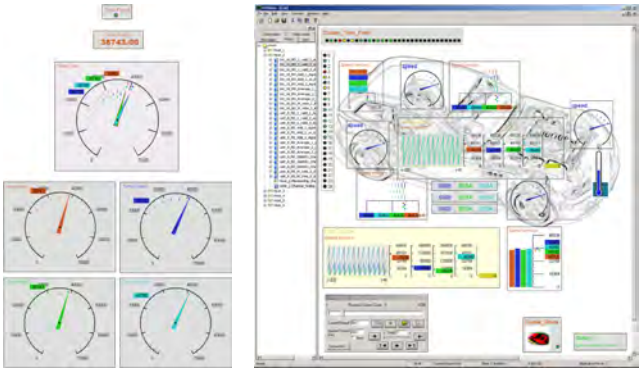


Figure 2.6.: Process visualization with levels of detail [Matković et al., 2002a]. The left-hand side of this figure shows virtual monitoring instruments at different levels of detail as suggested in [Matković et al., 2002a]. A screenshot of the system TTPView 3.0, which incorporates focus+context process visualization, is displayed on the right.

case, events are helpful in managing critical situations that might occur during the proceeding of some monitored process.

Considering Events in Intrusion Detection

The visualization of networks (e.g., social networks, communication networks) is an emerging field in visualization. Especially in the light of the continuously growing Internet, new techniques for visualizing network topologies or network performance have been developed in recent years. Another application of visualization becoming increasingly important is to support administrators in detecting intrusions into or misuse of network systems to ensure reliability of working environments (see e.g., [Erbacher et al., 2002], [Muniandy, 2001], [Scott et al., 2003]). A brief description of the event-related approach by Erbacher et al. will be given in the following.

In [Erbacher et al., 2002] it is described how intrusion and misuse detection in large scale networks can be supported by visual means. The authors have developed a visualization system that displays log files containing network-related messages generated by a monitored system [Erbacher, 2002]. In their idiom, events are the messages appearing in log files. The events describe, for instance, successful or rejected login attempts, different types of connections to a system as well as port scans.

To support intrusion and misuse detection, Erbacher et al. suggest the visualization of events with respect to a monitored system. In their visual display (see Figure 2.7), a monitored server system is represented by a central glyph, which represents the number of users and the load of the server. Events are represented by radially arranged lines of different appearance depicting the event types (e.g., port scan, ftp connection). The lines are arranged with respect to the remote host that is involved in the event. The host itself is represented by a small glyph at the end of a line. Additionally, the remoteness of the connecting host is encoded in the length of the drawn line. To preserve a certain

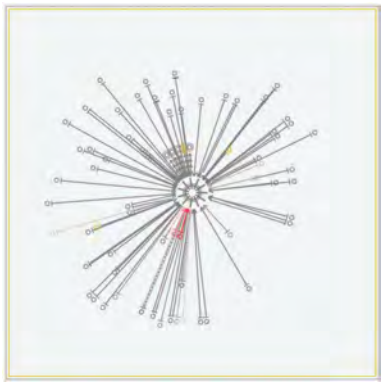


Figure 2.7.: Visually supporting intrusion detection [Erbacher et al., 2002].
The glyph in the center of the display represents the monitored system. Connections from remote hosts are depicted by radially arranged lines; hosts are shown as glyphs at the ends of the lines. Critical or suspicious activity is visualized by using red or yellow color, respectively.

history of connections that have been terminated, the respective lines are faded out gradually, rather than simply omitting them.

Whereas all drawings are usually done with a shade of gray, unexpected or suspicious activity (i.e., particular messages in log files) results in a change of color – red for critical activity, yellow for suspicious activity (see Figure 2.7). Hosts that try to open a privileged connection are colored in red; hosts that fail to respond turn yellow. Additionally, lines representing timed out connections or connections that failed the authentication procedure are shown in red. Connections that have been identified (by an external intrusion detection system) to be possibly intrusions are represented by lines and glyphs of even brighter red .

The authors show that highlighting of suspicious events helps administrators in observing network communication. Presenting colored (red or yellow) lines among gray lines attracts the attention of administrators to suspicious activity. Hence, actions can be taken quickly to counter network attacks from remote hosts. However, the events considered in this approach originate from a mixture of the authors' knowledge and an external intrusion detection system. Although users can visually recognize suspicious behavior, they cannot specify their insights as new events for later automatic highlighting.

Events in Clinical Visualization

In [Chittaro et al., 2003] a system is described that is dedicated to supporting physicians in the analysis of clinical data. Actually, the system developed by Chittaro et al. is used for visualizing hemodialysis-related time series data. These data contain several attributes (e.g., blood pressure) collected during hemodialysis sessions of different patients.

Chittaro et al. state two concurring aims that have to be achieved by their system. On the one hand, representing multivariate time-dependent data necessitates sophisticated visualization methods; on the other hand, the addressed users (i.e., physicians) are usually not familiar with complex visual representations. Therefore, well-known 3D bar charts are used as the basis for the visualization. To display multiple time series (i.e., the hemodialysis sessions), multiple 3D bar charts are created and arranged in a rectangular fashion. To support the visual analysis of the data, and to overcome the disadvantages of 3D displays (e.g., occlusion of information) a variety of specialized yet simple to operate interaction techniques (e.g., aggregation of dense bars or dynamic queries by means of a "water level" metaphor) are provided by the system.

What is special about the proposed system is that not only clinical data, but also clinical events are considered. The authors describe that events *... are related (...) to the intervention of the clinical staff during a session (...).* Events are not limited to certain interventions by clinical personal, they can also be unexpected actions taken by patients. However, events are not differentiated by some type. The only thing that matters is that something has happened, irrespective of the concrete type of happening. The considered events are loosely coupled with the clinical data by an indication of their occurrence time. This enables an integration of events in the visual display. To represent events, blue signs are displayed on top of 3D bars that represent a time step and a session for which an event occurred.

Figure 2.8 shows how events are marked with blue signs to highlight them among the rest of the visualization. Physicians analyzing the data can now easily recognize and identify unexpected situations that occurred during a hemodialysis session. If necessary, physicians can request further information about the happenings from the personal, or can consult the patients directly.

2.4.2. Other Event-Related Visualizations

In the previous section, visualization techniques that are more or less directly connected to the field of information visualization were described. This section will take a look on how event-related visual methods are applied in different scenarios.

Software Visualization and Events

Teaching algorithms is an essential task in the education of students in Computer Science. Another challenging task in Computer Science is to support programmers in developing flawless and stable software systems. Here it is necessary to provide helpful tools for debugging developed applications. Visual techniques have been used for years to support teachers and developers in accomplishing their tasks.

The visualization of software is an interdisciplinary field of research combining knowledge from education, software engineering, and visualization. Two principle approaches to visually representing software are known from literature [Demetrescu et al., 2002]. On the one hand, visualizations can be generated based on data states of software (data-driven approach). Alternatively, it is also possible to create visualizations based on events of interest (event-driven approach). In the following, the role of events in

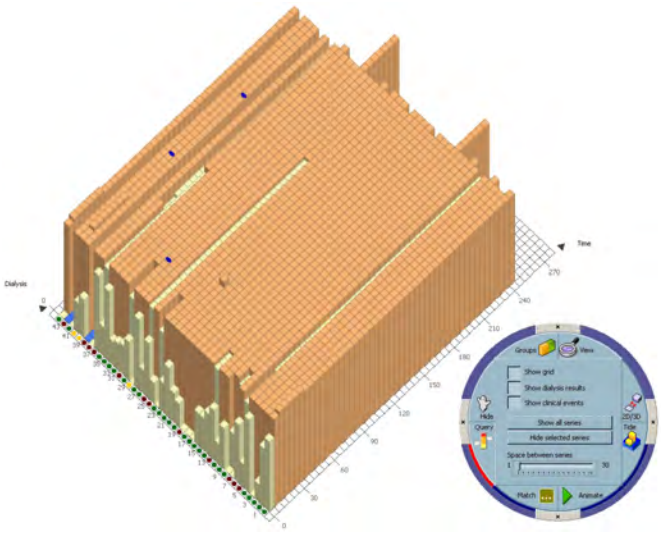


Figure 2.8.: Visualization of clinical data incorporating events [Chittaro et al., 2003]. Time-dependent data about hemodialysis sessions of many patients is represented by parallel 3D bar charts. Unexpected events are indicated by blue signs on top of particular bars to allow an easy recognition of the specific sessions and the time points of event occurrences.

algorithm animation and visual debugging is reviewed. In both scenarios, events are considered as happenings of interest. However, the precise definition of what constitutes an event of interest varies among the approaches known in software visualization literature. For more details on visualization of software in general, the interested reader is referred to [Stasko et al., 1998]; a comparison of data-driven and event-driven approaches can be found in [Demetrescu et al., 2002].

Algorithm animation Usually, algorithms are visualized by means of dynamic animations whose transitions are controlled by events of interest [Brown and Sedgewick, 1998]. The event-driven approach necessitates the identification of interesting events in the code of an algorithm (e.g., swapping of two items in a sorting algorithm). However, the *...precise definition of what constitutes an "interesting event" varies between authors* [Rößling, 2002]. The task of identifying events is obliged to the person who creates the animation. Since the event identification can be conducted at different levels of granularity, events can vary from simple assembler commands to entire higher language procedures [Roman and Cox, 1993]. Once interesting events are identified, they must be annotated manually with appropriate calls to an algorithm visualization system (see Figure 2.9). These calls and, in turn, the algorithm animation system are responsible for the creation of the visual representation and its animation.
Being intuitive and flexible (*...virtually any conceivable visual representation can be generated* [Demetrescu et al., 2002]) are the advantages of event-driven algo-

rithm animation. The created animations facilitate the understanding of algorithms, and thus simplify the task of educating students in Computer Science. It is stated as disadvantageous that animation creators must know the code and the animation system quite well to be able to identify and annotate all characteristic events of an algorithm appropriately.

```
int v[] = {3,5,2,9,6,4,1,8,0,7}, n=10, i, j;
void main(void) {
   bsort.SendAlgoEvt("Input",n,v);
   for (j=n; j>0; j--)
      for (i=1; i<j; i++)
         if (v[i] > v[i+1]) {
            int temp = v[i]; v[i] = v[i-1]; v[i-1] = temp;
            bsort.SendAlgoEvt("Exchange",i,i-1);
         }
}
```

Figure 2.9.: Event-driven creation of algorithm animations [Demetrescu et al., 2002]. The code snippet shows the Bubble-Sort algorithm annotated with calls (bold-italic) to an algorithm animation system. The calls are used to identify events of interest on whose occurrence animations of the visual representation shall be performed.

Visual debugging In visual debugging, events denote *...actions without duration that take place at specific points in time and change the state of a process* [Kranzlmüller et al., 1996]. If parallel and distributed systems are considered, a variety of processes and events have to be taken into account. For debugging purposes, so called *event graphs* can be extracted from trace files, which contain all events (usually procedure calls) that occurred during an execution of a considered system. Besides these so called primitive events, abstract events can be derived [Kunz et al., 1997]. For instance, a broadcast sent from one process to all other processes would consist of as many primitive events as processes are involved in the broadcast. To ease the understanding of program execution, these primitive events can be abstracted to one single broadcast event.
The event graph is represented visually by a space-time diagram that consists of stacked horizontal lines, one for each process involved (see Figure 2.10). Nodes on these lines represent events occurred in the respective process at a certain point in time. Since events in one process usually initiate events in other processes, directed lines between two nodes are used to represent this causality. Primitive events that constitute an abstract event can be omitted from the display. Instead, a visual indication specific to the derived abstract event is given. For instance, for a broadcast event, this indication is given by a vertical line that crosses all processes involved.
Space-time diagrams as presented previously facilitate the debugging of large software systems.

As a conclusion from the previous descriptions it can be stated that events are crucial in software visualization. Events are used to abstract from the behavior of

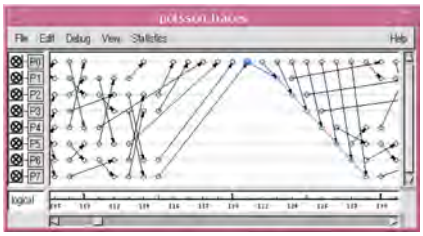


Figure 2.10.: Visual debugging with event graphs [Kranzlmüller, 2000].

single algorithms or whole software systems. This abstraction allows the creation of easily comprehensible visual representations that allow an effective analysis of software. Since the complexity of software systems and even the complexity of algorithms are steadily growing, supporting developers in debugging software systems and teachers in educating algorithms remains a challenging task (see e.g., [Knüpfer et al., 2005]).

Active Databases and Event Visualization

As described in Section 2.3, active databases integrate events into database management systems. Events of interest, conditions under which these events are relevant, and actions as a response to events can be described by means of ECA rules. These rules are not only useful to maintain data consistency, but can also be used to support the visualization of data stored in active databases.

One possibility is to update a visual representation automatically when changes to the data have occurred. In [Díaz et al., 1994] it is described that ECA rules are an eligible means for this purpose. A mechanism is presented that uses two ECA rules. One rule is used to notify a visualization system about changes to the database, irrespective of whether the change regards the data itself or the data schema. On receiving such notification the visualization updates itself to reflect the current changes. Since usually not all data are displayed, a second rule is used to inform the active database management system about the data portions currently shown. In this case, the visualization system acts as an external event generator. As response to such notifications the active RDBMS updates the former rule in a way that it only triggers visualization refreshes for data actually displayed.

In another more recent publication [Leissler et al., 2000], a coupling of active databases and VRML visualization is suggested to achieve automatic updates of represented data. Again ECA rules are used to trigger the notification. In this case, however, an external middle tier component is notified, rather than the visualization itself. This middle tier component allows connecting to it multiple visualizations, which are all kept up to date. The actual visualizations are described by means of VRML documents. All VRML documents are extended by an external sensor node that enables the VRML documents to listen to the middle tier component. When a notification is received at a sensor node, Java Script code is executed to change the content of the VRML documents according to the updated data in the database.

Whereas the previously presented approaches exploit events to update a visualization automatically, the work presented in [Coupaye et al., 1999] focuses on representing events in active RDBMS, more precisely, on visualizing executions of ECA rules. To help database administrators understand the implications of stored ECA rules, different basic views are provided, each of which is dedicated to different aspects of ECA rule execution. These aspects include occurrence of events, triggered reactions, or cascaded event triggering. Furthermore, the authors describe how these basic views are combined to a composite 3D view that conveys multiple aspects as follows (see Figure 2.11):

- Event instances are visualized as geometric objects aligned with respect to their occurrence time and priority.
- Reactions to an event are represented with the same type of geometric object as used for the triggering event. The reaction object is aligned with respect to the module in which the reaction is defined (differently colored grids).
- Cascaded triggering is visualized be means of lines connecting events and reactions.

This approach to visualizing ECA rules is helpful in analyzing effects caused by rule execution in active databases. Especially the capability of representing cascaded event triggering facilitates handling of heavily inter-depending rules. According to [Coupaye et al., 1999] the presented approach provides a general method to visualize not only ECA rules, but any kind of event-driven dynamic behavior.

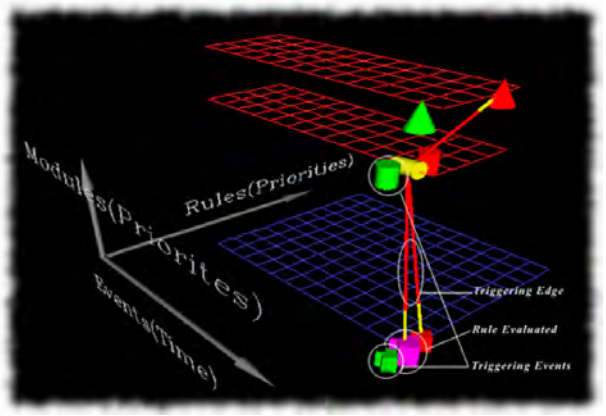


Figure 2.11.: Visualization of Event-Condition-Action rules [Coupaye et al., 1999]. The figure shows a simple example of ECA rule visualization. Events and Reactions are represented as small geometric objects aligned with respect to their occurrence time, priority, and modules they belong to. Cascaded triggering is visualized by lines connecting events and reactions.

The approaches reviewed in this section differ in the goals they try to achieve by combining visualization and active databases. On the one hand, active databases

are used as technical means to support dynamic visualization; on the other hand, visualization is used to support understanding of rules in active databases. This shows that potentially both areas – active databases and visualization – can benefit from each other.

Further Approaches

This section will show that not all visualizations related to events actually use event methodology. In fact, a variety of visualization approaches is specifically focused on only representing event data. These data contain information about events that have occurred in a certain observation space. This observation space usually includes at least one temporal dimension and may contain further spatial or abstract dimensions.

In [Gatalsky et al., 2004] events are understood as *...discrete and abrupt phenomena. They are most often considered as points in space.* Hence, the authors refer to data that contain information about observed earthquakes or traffic incidents as events. In the particular case of earthquakes, the data is defined by one temporal and two spatial dimensions as well as further attributes like intensity of an earthquake. Gatalsky et al. utilize a visualization technique known as space-time cube to represent the spatio-temporal aspects of event data. For doing so, events are represented by small circles aligned in a 3D presentation space where one axis encodes time and two axes are used to represent the spatial dimensions. An additional attribute can be mapped to the size of the circles. From a more abstract point of view the presented visualization technique can be considered a 3D Scatter Plot. As such, the space-time cube is capable of visualizing not only specific event data, but any kind of spatio-temporal data.

A second example regards a larger research project that focuses on analyzing data stemming from network and system management. In [Ma and Hellerstein, 1999] events are considered to be certain network- or system-related occurrences (e.g., opening or closing of internet connections). External tools are used to detect such events on an observed system. Collections of events are given in form of log files, which contain not only the events themselves, but also attributes related to the events. The data contained in the log files is called event data. Although different event types differ in the attributes associated with them, the attributes time of occurrence, event origin, and event type are common to all events. Hence, the data can be characterized as multivariate time-dependent data with a larger number of categorical attributes (e.g., host names). The visualization system developed for analyzing these event data is based on Scatter Plots [Taylor et al., 2000]. To support the interactive data analysis, the plots can be used to select portions of the data for further detailed analysis dynamically. Another focus set by the authors concerns the mining of certain patterns in event data. In [Hellerstein et al., 2002] patterns like event bursts or periodic event occurrences are introduced and sophisticated algorithms for detecting such patterns are described. How the visualization and the mining part are combined is described in detail in [Ma et al., 2002].

Both of the previously described approaches are related to events. However, events are not an essential part of the visualization process; an event-reaction schema is not used. The events are merely points of an observation space, i.e., the data are the

events. In fact, standard visualization techniques (3D or 2D Scatter Plots) are used to represent event data. Hence, the proposed systems are not limited to event data, but can be applied to visualize any time-dependent multivariate dataset. As such, the approaches differ from the model developed in this thesis, which aims at incorporating an event-reaction schema to guide users to relevant parts of the represented data.

Chapter Summary

In this chapter, basic notions of visualization, relational databases, and event methodology have been explained. The main part of this chapter was dedicated to reviewing the state of the art of event-related visualization approaches. The previous paragraphs show that a variety of such approaches are known in literature. Few of them emphasize on events (e.g., [Reinders, 2001]) to improve visualization. In most case, events are more or less understood as add-ons (e.g., [Matković et al., 2002a]). In some cases it even seems that the potential of integrating events into the visualization is underestimated. Nonetheless, the diversity of scenarios making use of events suggests that it is useful to combine event-based methodology and visualization methods. Therefore, the goal of this thesis is to develop a general event-based visualization model and to describe how events can be successfully integrated using the developed model.

3. A Classification Schema for Event-Related Visualizations

The approaches presented in the previous chapter document the diversity of couplings of visualization with event-related methodology. This diversity makes it difficult to assess and evaluate the techniques known from literature. Due to the lack of a classification for event-related visualizations, potential users have difficulties in deciding whether a certain approach is applicable for their particular visualization problem. Therefore, a classification schema for event-related visualization methods will be developed in this chapter. The focus is set on the coupling of events with visualization, rather than on visualization specific aspects[1]. The developed classification schema does not only allow a categorization of the reviewed event-related visualization approaches, but also serves as a basis for identifying unsolved problems in visualization and deriving goals of this thesis in Section 4.1.

3.1. Classification Criteria

When developing a classification schema for event-related visualizations, the first step is to figure out expressive classification criteria. The classification criteria that have been identified consider the *event complexity*, the *event specification*, the *event integration*, the *temporal characteristics of data*, the *temporal characteristics of visualization*, and last but not least the *visual event representation*. All these criteria are described in detail in the following sections.

Event Complexity

The possible complexity of events has much impact on the expressiveness of what occurrences can be described by means of events. The event complexity also determines the applicability of event-based visualization approaches. With regard to event complexity, three kinds of events can be differentiated:

- Simple events,
- Composite events, and
- Mined events.

[1] A variety of classifications of visualization techniques can be found in literature (e.g., [Chi, 2000], [Card and Mackinlay, 1997]).

Simple events are based on ordinary comparison operations like $=$, $<$, or $>$. These operations can be realized as inter-comparison between values of a dataset or as comparison between data values and user defined values. Usually, such operations are applied in monitoring scenarios (e.g., [Matković et al., 2002a]). In such scenarios, events can be considered if, for instance, the value of one attribute equals the value of another attribute (inter-comparison). Furthermore, threshold exceeds could be of interest. In that case, an attribute's value is compared to a user defined threshold value. Apparently, simple events are sufficient for describing basic occurrences of interest.

More sophisticated visualization tasks can be accomplished by more complex events. This implies that the expressiveness of events must be increased. One possibility to do so is to create events by eligibly combining simple events (e.g., [Kranzlmüller et al., 1996]). Usually, logical operators ($\wedge$, $\vee$, or $\neg$) are used to create so called composite events. This enables scenarios where multiple comparison operations must hold before a happening is considered an event. However, the increase in expressiveness also means an increase in the effort required for specifying events.

Another kind of complex events with even higher expressiveness are mined events. These are events that cannot be described by comparisons or combinations of comparisons. Quite the contrary, mined events require sophisticated algorithms for detecting them from the data (e.g., [Reinders, 2001]). This implies that in most cases only a limited number of mined events are supported by a visualization. Usually, mined events are applied in scenarios where very specific visualization tasks must be accomplished like, for instance, in visual data mining (see [Keim, 2002]) or graph analysis (e.g., [Abello et al., 2002]).

Event Specification

The question of who is in charge of specifying events is raised when evaluating the flexibility of event-based visualizations. Two possible answers exist for this question (see Figure 3.1): Events can be specified either by the:

- Visualization designer, or the
- Visualization user.

If visualization designers specify events, it is possible for them to consider not only simple or composite events, but also very complex mined events (e.g., [Reinders, 2001]). Designers are able to integrate data mining methods, which are capable of finding correlations deeply hidden in the data. Such visualizations are capable of revealing and communicating complex relations in the data. However, the visualization designer must know the potential interests of users and the characteristics of the data quite well to be able to integrate all events that are necessary for understanding the data. This is usually not the case, and only a limited subset of potentially interesting events is considered. Thus, visualizations integrating only designer specified events are limited in the number of visualization task that can be accomplished. Due to the high degree of specialization of the mining algorithms used, there is also a limitation regarding the data that can be processed.

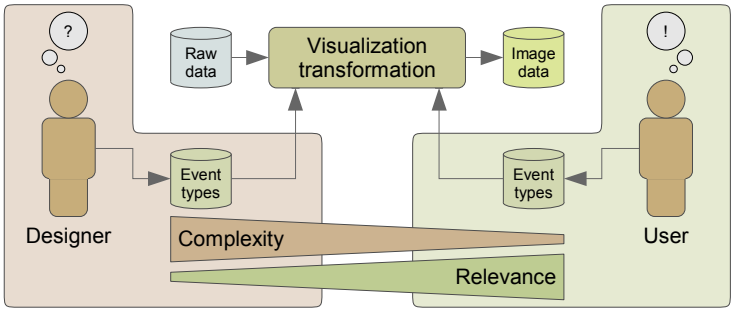


Figure 3.1.: Classification criterion – event specification.
The figure depicts the implications when event specification is done by the visualization designer or the visualization user. The designer is able to describe complex events that are usually kept very general. Their relevance is arguable with respect to specific users. On the other hand, if users are in charge, they can specify whatever events are relevant for them. However, the events will be of limited complexity.

On the other hand, users of a visualization know the events they are interested in. So it makes sense to integrate users into the event specification process. If users are in the position to define the events they are interested in, the number of possible events is virtually unlimited. Of course, in a real-world system the number of possible events is limited by the expressiveness of the event definition method. For simple events, it is easy for users to define comparison operations and values to compare to. Composite events are more difficult to specify for common users of visualizations. Therefore, it is necessary to support users in the event specification. Depending on the users' knowledge and experience, this can be achieved by means of visual languages. More complex mined events cannot be defined by users directly, because programming skills are required for accomplishing this task.

Event Integration

Another event-related classification criterion regards how events are integrated into the visualization process. If events have been specified by either means, the question is where do actual event instances originate from. The source of actual event instances is a crucial criterion that has much impact on the architecture of a visualization system. To characterize event-related visualizations regarding event integration, the following distinction (see Figure 3.2) is made:

- Event instances are given a priori only,
- Event instances and data are given separately, or
- Event instances are detected from the data.

In a variety of scenarios, events are collected by external tools or sensory components. These event occurrences are then stored, for instance, in log files (e.g., [Erbacher et al.,

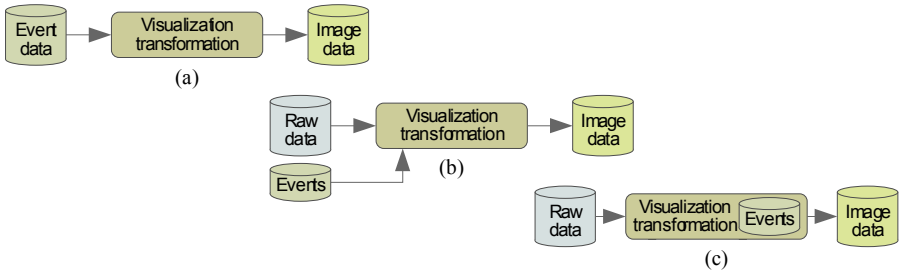


Figure 3.2.: Classification criterion – event integration.
The figure depicts the different possibilities to integrate events into the visualization process. (a) Events are given a priori (as event data); (b) Events and data are given separately; (c) Events are extracted from the data.

2002]) or in so called event data (e.g., [Gatalsky et al., 2004]). In this case, log files as well as event data can be understood as time-dependent datasets. These data are usually given a priori and can be processed through the visualization pipeline as usual; an additional event extraction step is not required.

It is also possible that data and event instances are given separately. For example, this is the case if time-dependent data are considered, and certain events have been observed during the collection of the data (e.g., [Chittaro et al., 2003]). This means that, besides processing the data, the information about events must be integrated during the creation of the visual representation.

If arbitrary data shall be visualized in an event-based manner, it is necessary to extract events from the data, i.e., the events considered originate from within the visualization process. Hence, the extraction of events is an essential step in the creation of a visual representation of some data (e.g., [Reinders, 2001]). This high degree of integration of events and visualization allows sophisticated visual displays.

As a matter of fact, events originating from external sources (i.e., event data) can be the input to a visualization system with an event extraction step. This means that new events can be extracted even from data already consisting of events.

Visual Event Representation

The classification criterion described next regards how events are considered in the visual representation. This criterion is the most interesting one from the visualization point of view. There are two principle ways of representing events: They can be visualized:

- Implicitly, or
- Explicitly.

By implicit event representation it is meant that events have impact on how the data are processed through the steps of the visualization pipeline; events control the outcome

of the visualization transformation. How this is actually realized varies from technique to technique. Possible ways are for example to enhance a visualization with visual cues representing events (e.g., [Chittaro et al., 2003]) or to adjust the visualization depending on occurred events (e.g., [Matković et al., 2002a]). If the number of events is large, it might turn out difficult to represent events implicitly.

In cases where a large number of events must be represented visually, an explicit event representation is more sufficient. An explicit visual representation of events is realized by a specialized visualization technique using a dedicated drawing space (e.g., [Reinders, 2001]); the event representation is separated from the data visualization. If users are only interested in events, the data visualization can even be omitted. By means of explicit event visualization it is possible to show events as well as relations between events. However, the direct connection with the data is cut off and must be restored by users mentally.

Temporal Characteristics of Data

Events are always related to alteration, that is, events can only occur if something changes. If events shall be considered in visualization, the data must be somehow connected to time. From the field of temporal databases it is known that data can be related to two temporal domains:

- Internal time $\mathfrak{T}_i$ and
- External time $\mathfrak{T}_e$.

The internal time is the time inherent to the data model, whereas the external time is extrinsic to the data model[2]. Depending on the number of temporal elements in $\mathfrak{T}_i$ and $\mathfrak{T}_e$, different temporal characteristics of datasets can be identified:

Static non-temporal data If $\mathfrak{T}_i$ as well as $\mathfrak{T}_e$ contain each no more than one temporal primitive, then the data are completely independent from time.

Static temporal data If $\mathfrak{T}_i$ contains multiple temporal primitives, whereas $\mathfrak{T}_e$ contains only one element, then the data are time-dependent. The data can be considered a historical view of how the data looked like at specific elements of $\mathfrak{T}_i$. Common time series are an example for this type of data.

Dynamic non-temporal data If $\mathfrak{T}_i$ contains only one temporal primitive and $\mathfrak{T}_e$ comprises multiple temporal primitives, then the data are dependent on external time. In other words, the data change over time. Only the current state of the data model is preserved at each element of $\mathfrak{T}_e$. Such data could be, for instance, a document database of a library where newly acquired publications are added occasionally.

[2]In temporal database terminology, the notion *valid time* is used for the time inherent to the data model, and the notion *transaction time* is used for external time [Snodgrass and Ahn, 1985].

Dynamic temporal data If $\mathfrak{T}_i$ as well as $\mathfrak{T}_e$ contain multiple temporal primitives, then the data are considered to be bi-temporally dependent, i.e., the data contain attributes depending on time and the actual state of a dataset itself varies over time. In this case, $\mathfrak{T}_i$ and $\mathfrak{T}_e$ are usually linked and can be transformed to each other by some mapping function. Examples for such data could be health data or climate data that contain measures depending on time (e.g., daily number of cases of influenza or daily average temperature), and that are updated after every 24 hours with data of the passed day.

A figurative representation of possible temporal characteristics of data is given in Figure 3.3, which was inspired by [Steiner, 1998].

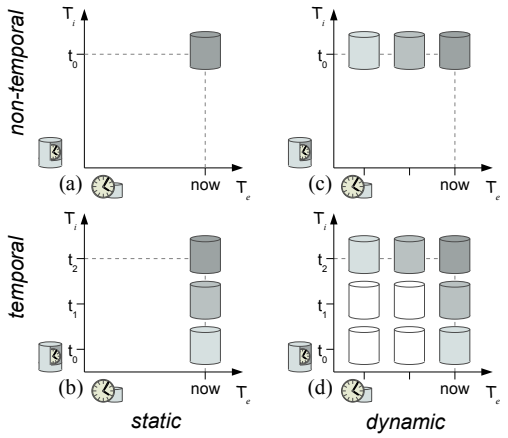


Figure 3.3.: Classification criterion – temporal characteristics of data.
A dataset can depend on internal time $\mathfrak{T}_i$ and/or external time $\mathfrak{T}_e$. Depending on the number of temporal primitives in $\mathfrak{T}_i$ and $\mathfrak{T}_e$, different types of data can be identified. (a) A dataset completely independent from time (static non-temporal data). (b) A dataset with an inherent time dependency (static temporal data). (c) A dataset that does not contain time-dependent attributes, but that changes over time (dynamic non-temporal data). (d) A dataset that depends on internal and external time (dynamic temporal data).

Temporal Characteristics of Event Representations

Not only the temporal characteristics of the data are of interest, but also the temporal behavior of visual representations of events. In this regard, event visualizations can be characterized as:

- Static or
- Dynamic.

Event visualizations where no automatic alteration of the visual representation occurs are considered to be static (e.g., [Chittaro et al., 2003]). The relevant event information is represented within a single visual representation.

On the contrary, dynamic event visualizations base on "world" time, i.e., they utilize time as an additional dimension for encoding information (e.g., [Matković et al., 2002a]). For creating dynamic event visualizations, multiple visual representations of data and/or events, or portions of those are created and displayed one after another, for instance, as slide shows or animations. Obviously, such representations can excellently convey temporal characteristics of events. However, for users it is often difficult to remember all information they have been provided with during the progression of a dynamic representation [Tversky et al., 2002]. This holds true especially for dynamic representations that take a long time.

It must be mentioned that it is commonly the case that interaction techniques can be used to alter visualization parameters to create different views of the shown events. Since interaction is not automatic, but caused by users, the presence or absence of interaction techniques has no influence on whether an event representation is categorized as static or dynamic.

3.2. Categorization of the Reviewed Approaches

Before classifying the approaches reviewed in the previous sections, it should be mentioned that, although in principle all classification criteria are orthogonal to each other, practically there exists one dependency. This dependency is between the event complexity and the event specification, because it is – even though theoretically possible – in reality very unlikely that visualization users will be in the position to specify complex mined events.

The introduced criteria can now be compiled to a classification schema for event-related visualization approaches. The classification schema as well as the previously described event-related visualization approaches are depicted in Table 3.1. In table 3.1, the reviewed approaches are listed by numbers that correspond to the following enumeration, which provides a brief recapitulation of the previous sections:

1. **Feature-Based Flow Visualization [Reinders, 2001]:** Events extracted from time-dependent flow data are represented as event graph.
2. **Process Visualization [Matković et al., 2002a]:** Events are used to provide virtual monitoring instruments at different levels of detail.
3. **Visualization for Intrusion Detection [Erbacher et al., 2002]:** Suspicious network-related events are highlighted to support intrusion and misuse detection in computer networks.
4. **Visualization of Clinical Data [Chittaro et al., 2003]:** The visualization of clinical data is enhanced with visual indications of treatment-related events.

5. **Event-Based Algorithm Visualization [Brown and Sedgewick, 1998]:** Events are used to create visual representations of algorithms.
6. **Visual Debugging [Kranzlmüller et al., 1996]:** Events collected during the executing of distributed programs are visualized by means of space-time diagrams.
7. **Visualization of ECA Rules [Coupaye et al., 1999]:** Event-condition-action rules are visualized to convey their implications in active databases.
8. **Event Visualization Using Space-Time Cubes [Gatalsky et al., 2004]:** Geographical event data are visualized by means of space-time cubes.
9. **EventBrowser [Ma and Hellerstein, 1999]:** Events occurring in network and system management are visualized by means of Scatter Plots.

Event Complexity	Simple Events	Composite Events	Mined Events	
	2. 3. 4. 5. 8.	6. 7.	1. 9.	
Event Specification	Visualization Designer	Visualization User		
	1. 2. 3. 4. 5. 6. 7. 8. 9.			
Event Integration	Event Data Only	Events and Input Data separately	Events Detected from Data	
	3. 6. 7. 8.	4. 5.	1. 2. 9.	
Visual Event Representation	Implicit	Explicit		
	2. 4. 5.	1. 3. 6. 7. 8.		
Temporal Characteristics of Data	Static Non-Temporal Data	Static Temporal Data	Dynamic Non-Temporal Data	Dynamic Temporal Data
		1. 3. 4. 5. 6. 7. 8. 9.	2.	
Temporal Characteristics of Event Representations	Static Representation	Dynamic Representation		
	1. 4. 6. 7. 8. 9.	2. 3. 5.		

Table 3.1.: Classification of event-related visualization approaches.
The numbers used denote the different reviewed approaches. A recapitulative list of these approaches and respective numbers are given on page 37.

Table 3.1 illustrates how the different approaches fit into the proposed classification schema. Without going into too much detail it can be seen from Table 3.1 that:

- Most techniques consider only simple events.
- Users are not involved in the process of specifying events of interest.
- Only one third of the reviewed approaches actually detect events from some data.
- There is no general preference for using either implicit or explicit event representations.
- Almost every approach visualizes data with an inherent temporal dimension. Only the approach by [Matković et al., 2002a] considers data that changes over time (i.e., dynamic data).
- Static and dynamic visualization techniques are used for representing events.

This list documents the strengths and weaknesses of current event-related visualizations. An important aspect worth to be mentioned here regards the fact that none of the classified approaches covers multiple characteristics per classification criterion. In this sense, all reviewed approaches are specific to a particular visualization task. A more detailed analysis of problems of the reviewed event-related visualizations in the context of current challenges in visualization research will be given in the first section of the next chapter.

4. A Novel Generalized Approach to Event-Based Visualization

The previous chapters have shown that events are helpful tools in Computer Science. Events have also proved their eligibility in the field of visualization. However, current event-related visualizations do not exploit the full potential of event-based visualization. Therefore, this chapter is dedicated to the development of a novel generalized approach to event-based visualization. In a first section of this chapter, challenges and unsolved problems in current visualization research are identified and the goals of this thesis are derived. Thereafter, the general event-based visualization model is introduced by illustrating its principle components informally. Based on that, a formalization of the suggested concepts is developed. The main aspects of event-based visualization – the event specification, the event detection, and the event presentation – are described in detail in separate sections of this chapter.

4.1. Challenges, Unsolved Problems, and Goals

Visualization has proved to be an effective tool for supporting data analysis tasks in various application fields. Still, there are many unanswered questions. Therefore, unsolved problems in visualization research that can possibly be alleviated using event-based visualization will be identified in this section.

Coping with ever increasing volumes of data is a task that is challenging for every visualization technique. Usually, visual representations of large datasets are cluttered and overcrowded, and thus difficult to understand. Besides using approaches like overview+detail and focus+context, this problem can be dealt with by visualizing only relevant parts of the data. Extracting interesting events is one possibility to focus on relevant data portions. Among the reviewed event-related approaches only the work of [Reinders, 2001] actually takes advantage of events to reduce the amount of information to be represented. However, Reinders' description of events is specific to only one particular visualization problem, and hence is not generally applicable.

Another problem related to data is that in a dynamic world data are usually subject to steady change. Yet, the importance of dynamically changing data is underrepresented in current visualizations. Even if highly dynamic data are considered, they are often visualized in an off-line manner only (e.g., [Erbacher et al., 2002]). When facing dynamic data, it would be a better solution to automatically identify interesting changes in the data, and to emphasize such changes in the visual representation.

The specialization of visualization techniques to only one particular visualization task remains another unsolved problem. Most of the known techniques visualize only

very specific data, not only in terms of the data characteristics, but also regarding the data format. As can be seen from the approaches reviewed in Chapter 2, data are mostly read from proprietary formatted files. Even if data are given in relational databases, only file dumps of these data are accessed when creating visual representations; a direct connection to relational databases is considered as an extension only (e.g., by [Erbacher, 2002]). On the other hand, most of today's datasets are stored in relational databases, since they offer rich facilities for managing and accessing very large volumes of data. The advent of active databases [Paton, 1999], which incorporate event-based mechanisms for managing relational data more efficiently and more flexibly, will surely cause relational databases to become ubiquitous on every computing device. The question that arises is: Could event-based methodology bring similar improvements to visualization? The works of [Díaz et al., 1994] and [Leissler et al., 2000] have shown, although by simple means, that a combination of active databases and visualization technology bears great potential to support the creation of visual representations.

The most important unsolved problem in visualization that remains even if current event-related visualization approaches are considered, is the low degree of involvement of users in the visualization process. In most cases, visualization designers determine what should be of interest and users are restricted to only viewing the visual representations. This limits not only the expressiveness of a visualization approach, but also its applicability for different or even similar visualization tasks. As a matter of fact, it is the users who know their interests best, and hence, they should be integrated more tightly into the creation of visual representations. Events that can be defined by users are one possibility to achieve this demand. Despite the fact that users definitely need to be supported in specifying their interests as events and that user-defined events usually cannot be as complex as events specified by visualization designers, great potential is expected from a higher degree of user involvement.

In conclusion, the major problems of visualization that are addressed in this thesis are **decrease of effectiveness** (overcrowded and cluttered visual representations), **lack of flexibility** (limited applicability of visualization techniques), and **lack of relevance in visual representations** (no consideration of user interests). The goals that infer from these problems are:

1. Increase effectiveness of visual representations,
2. Increase flexibility of visualization, and
3. Increase relevance in visual representations.

These goals shall be achieved by taking the users' needs and interests into account when creating visualizations of (abstract) data. Event-based techniques have proved their eligibility in many fields of Computer Science. From the previous sections it can be seen that also a variety of visualization approaches make use of event-related concepts. This gives evidence that event-based methods can potentially improve the visualization, and furthermore, that event-based visualizations are useful in many application fields. However, the proposed approaches are limited in their capabilities, and moreover, are always specific to only one visualization problem.

To achieve the goals of this thesis, a **general model of event-based visualization** is required. In the next sections, such a model is introduced and the following essential issues are discussed in detail:

1. Event specification,
2. Event detection, and
3. Event representation.

It is the aim of this work to present a general approach to event-based visualization that should be easy to adapt to specific visualization problems, and should be flexible enough to cover most of the aspects shown in Table 3.1. Some of the aspects are particularly of interest in this thesis. A major concern is the integration of users into the visualization process. On the other hand, some aspects are less relevant for this thesis: Neither are mined events considered, nor will events be integrated as event data. Table 4.1 depicts the aspects that are in the focus of this thesis.

Event Complexity	Simple Events	Composite Events	Mined Events	
Event Specification	Visualization Designer	Visualization User		
Event Integration	Event Data Only	Events and Input Data separately	Events Detected from Data	
Visual Event Representation	Implicit	Explicit		
Temporal Characteristics of Data	Static Non-Temporal Data	Static Temporal Data	Dynamic Non-Temporal Data	Dynamic Temporal Data
Temporal Characteristics of Event Representations	Static Representation	Dynamic Representation		

Table 4.1.: Aspects of event-based visualization addressed in this thesis.
The goal of this thesis is to develop a general model of event-based visualization that covers the characteristics shown in green. A special focus is set on the integration of users into the visualization process (red). Less-relevant characteristics are depicted in gray.

4.2. An Event-Based Model of Visualization

The reason for developing a general event-based model of visualization is to take advantage of event-based methodology. A major goal was to combine all necessary event-related aspects with the visualization pipeline (see Section 2.1, [dos Santos and Brodlie, 2004], or [Haber and McNabb, 1990]). By doing so, the commonly accepted procedure of creating visual representations can be coupled with advantages of event-related concepts.

Before introducing the model in detail, it makes sense to illustrate the general idea of the model, and to clearly state what exactly is meant by *event*. The basic idea of event-based visualization is similar to any event-related approach: Automatically perform certain actions on occurrence of events of interest. Speaking in terms of event-based visualization, this means that events of interest that have been detected in the data are supposed to be automatically highlighted in the visual representations, i.e. the visual representations focus on interests matched in the data.

In this thesis, an event is considered if conditions that users are interested in are suddenly all met in the data. In this sense, events can be considered special portions of the data that are of particular interest, because they match with some interest expressed by users. In the context of relational databases, the data portions addressed can be of different types, for instance, data tuples, attributes, or collections of tuples. Data portions of equal types are collected in so called *event domains*. A specific interest with respect to the elements of an event domain is expressed as a condition. Since a condition describes what makes a data portion interesting, it can be understood as an *event type*. An *event instance* is considered for those portions of the data for which the condition is satisfied, i.e., if something interesting is contained in the data. An event instance is always bound to an event type and to the element of an event domain for which the condition holds. An event instance may also be associated with some *event parameters*. In conclusion, the notions of event domains, event types, and event instances build up the basis for the development of a general event-based visualization model.

Three major issues must be investigated to achieve event-based visualization – *event specification*, *event detection*, and *event representation*. Figure 4.1 illustrates how these fundamental components are incorporated into the novel general model of event-based visualization. For now, only brief descriptions of the model's main components will be given.

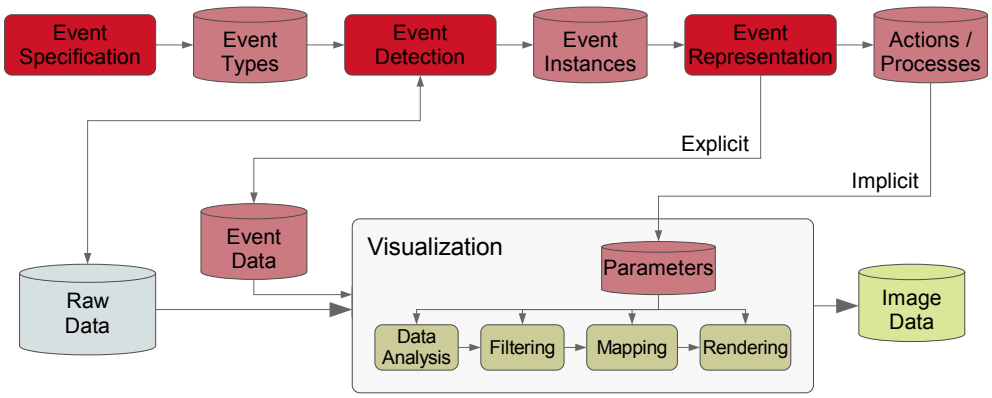


Figure 4.1.: General model of event-based visualization.
This figure depicts the major components of event-based visualization – event specification, event detection, and event representation – attached to the classic visualization pipeline.

Event specification The task of the event specification is to compile event types that are or might be of interest to visualization users. The event specification necessitates a formal foundation to allow a later detection of event instances. Furthermore, intuitive tools must be developed to allow users with different experience to specify their interests as event types.

Event detection The goal of the event detection is to find actual event instances of previously specified event types in the data to be visualized. To accomplish this goal, it is necessary to evaluate the conditions constituted in the event types. If complex event types are allowed, and if highly dynamic and/or large datasets are considered, sophisticated methods are required for detecting event instances efficiently.

Event representation Taking detected event instances into account when generating visual representations is the third major aspect of event-based visualization. Three goals were identified that must be achieved to gain benefit from event-based visualization. First, users must be made aware of the fact that one or more event instances (i.e., something interesting) have been found. Secondly, event instances have to be highlighted among the rest of the visual representation. This means that those parts of the display that represent interesting portions of the data have to become visually more present. The third goal is to visually communicate event types, i.e. to communicate what makes a data portion interesting.
The benefit that can be generated by using event-based visualization is mainly determined by the way in which events are actually represented. According to the classification schema developed in Chapter 3, events can be represented either implicitly or explicitly. In the implicit case, the event representation is realized by means of actions and/or processes that control the visualization parameters, and hence the outcome of the visualization pipeline. Actions and/or processes are bound to event types and are invoked with respect to the data portions for which event instances occurred. For the case of explicit event representation, detected event instances are mapped to a new dataset, which is processed through the visualization pipeline as usual.

The developed model can be applied to visualize arbitrary data. Originally, event-based visualization addresses dynamic datasets, i.e., data that change over time. An *event-based visualization session* for dynamic data can be described as an extension to the classic visualization session (see Section 2.1). The five steps in an event-based visualization session are:

0. In a preprocess, users describe their interests by means of event types.
1. Event instances are detected in the dataset to be visualized.
2. Event instances are represented
   a) Implicitly and/or
   b) Explicitly.

3. An interactive visual representation of data/events is presented.

4. On dynamic changes to the data go to step 1.

This general procedure will be clarified by an example. It is assumed that a physician visualizes a dataset that contains the daily number of cases of different diseases. Increases and decreases of different diseases can be read from charts that have been generated by a suitable visualization technique. Moreover, there is a special condition the physician is interested in: If the number of new cases of influenza increases for three successive days and the absolute number of cases of influenza exceeds 300, then this might be an indication of a possible wave of influenza. In an event-based visualization session, the physician would conduct the following steps:

(0) The physician specifies the condition of interest as an event type.

(1) The event detection tries to find actual instances of that event type. If no event instances are detected in the data, the charts are presented as normal.

(2.a) If an event is detected, the visual representation is adjusted to visually highlight the special situation. The charts are enhanced with markers, and color scales are altered to emphasize the data record for which the event has been detected.

(3) The physician can quickly and easily recognize the importance of the current health situation from the interactive visualization. This enables the physician to take steps to prevent a possible epidemic of influenza: Precaution measures can be initiated or the stock of vaccination medicine can be refilled.

(4) In case of changes to the dataset (e.g., official data are fetched from health care authorities) the event detection step is rerun and the visual representation is updated.

By the previous example, the usefulness of event-based visualization for dynamic datasets becomes clear. However, event-based visualization is not limited to dynamic data. Even static data can be represented by means of the proposed event-based approach. Since static data do not change over time, step 4 can be omitted. This means that events are simply detected only once (when the data are read for visualization), and only one visual representation is created. Even in this case, the resulting visual representation benefits from the highlighting of relevant information and the hiding of less-relevant information.

Another possibility to visualize static data with the proposed event-based approach opens up if the data contain an inherent temporal dimension $\mathfrak{T}_i$. An event-based visualization of such data can be realized by simulating a dynamically changing dataset. For doing so, a mapping must be defined that associates each element of the internal time $\mathfrak{T}_i$ with an element of the visualization time (i.e., external time $\mathfrak{T}_e$). If such a mapping can be found, the data can be made artificially dynamic, and the steps 0 to 4 can be performed as described. In this sense, the proposed model is general enough to be able to handle data with arbitrary temporal characteristics.

As conclusion of this section, Figure 4.2 shows a comparative view of the classic visualization approach and the proposed general event-based visualization approach.

When using the classic approach, all users are provided with the same visual representation, although they are interested in different aspects of the data (denoted by different colors). By following the event-based approach, it is possible to generate individually adjusted visualizations. It becomes clear by Figure 4.2 that the integration of the user into the process of creating visual representations of data helps to increase the relevance in the resulting images.

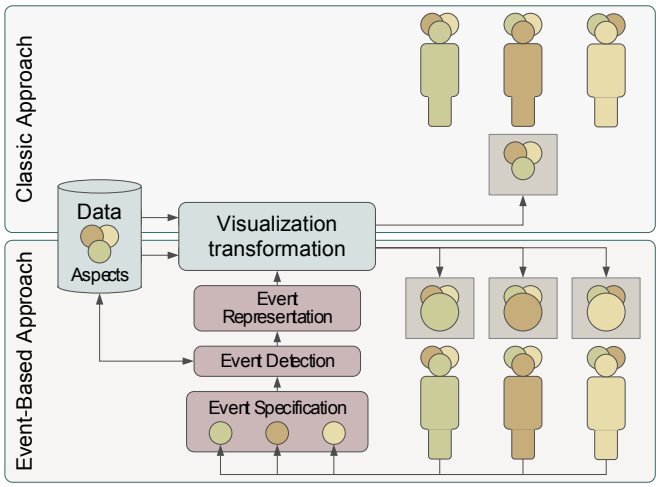


Figure 4.2.: Comparison of classic and event-based visualization.
The figure depicts users who are interested in different aspects (denoted by color) of the data to be visualized. Using the classic approach results in a standard representation. The event-based approach integrates users into the process of creating a visual representation by enabling them to specify their interests via event types. As a consequence, the relevance in the resulting images is increased.

Even though the brief description given in this section is sufficient for the introduction of the model of event-based visualization, all introduced issues must be examined in detail to substantiate the model. These examinations are carried on in the following sections.

4.3. Formal Description of Event-Based Visualization

In this section, the previously introduced model will be formally described. First, it is necessary to define event types and event instances as well as their frame of reference on an abstract level. Then, a visualization transformation can be defined that considers event instances so as to achieve event-based visualization.

To describe the frame of reference in which events occur, the notion of event domains is introduced. An *event domain*

$$ED$$

contains entities with respect to which event types can be specified. Such entities can be, for instance, tuples or attributes of a relational dataset, or simply time points. It is assumed that event instances occur with regard to entities of an event domain.

As indicated earlier, a distinction between event types and event instances is generally required in event-based systems. In event-based visualization, an *event type*

$$et \in ET$$

is used to express a concrete interest regarding the entities of an event domain. Here, ET denotes the set of all possible event types. To differentiate event types that address different event domains, the notion of abstract event types is introduced. The *abstract event type* for an event domain ED is defined as

$$\widehat{ED} \subseteq ET$$

where $\widehat{ED}$ comprises only those event types of ET that can be evaluated regarding the elements of ED. In other words, an abstract event type is a notation to associate an event domain with a set of compatible event types.

Actual event instances can be defined as follows. Assumed $et \in \widehat{ED}$ is an event type in some abstract event type $\widehat{ED}$ and an entity $ed \in ED$ conforms to the interest expressed as et, then the triple

$$e = (ed, et, EP) \in \mathcal{E}$$

denotes an *event instance* (or short *event*) of type et. The set of all possible event instances is denoted by $\mathcal{E}$. The set EP denotes *event parameters* that can be assigned to an event instance. As such, event instances establish a connection between interests (i.e., the event type) and concrete entities of some event domain. The introduced notions and their relations are illustrated in Figure 4.3.

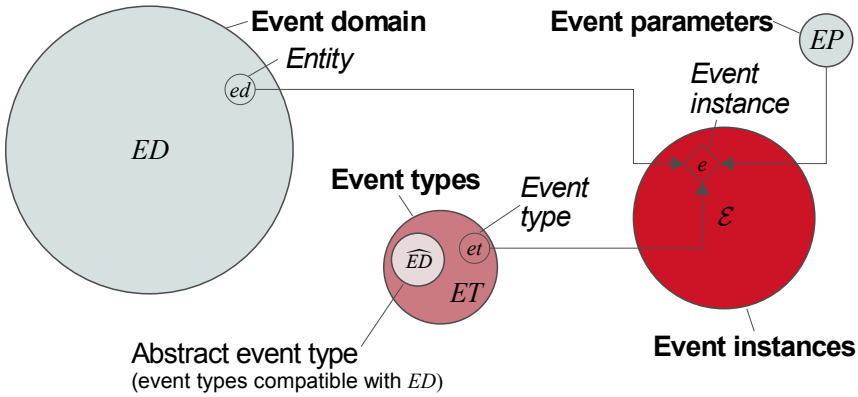


Figure 4.3.: Illustration of the notions event domain, event type, and event instance.

The notions of event domain, event type, and event instance are used to define event-based visualization formally as follows. The detection of actual instances of some event types with respect to a dataset can be modeled as a mapping from $\mathfrak{P}(ET)$ and $\mathcal{D}$ (the set of datasets) to $\mathfrak{P}(\mathcal{E})$ ($\mathfrak{P}$ denotes powersets)

$$\texttt{detect} : \mathfrak{P}(ET) \times \mathcal{D} \rightarrow \mathfrak{P}(\mathcal{E}).$$

Visualization in general can be modeled as a mapping from $\mathcal{D}$ (the set of datasets) and $\mathcal{P}$ (the set of visualization parameters) to $\mathcal{V}$ (the set of visual representations)

$$\texttt{vis} : \mathcal{D} \times \mathcal{P} \rightarrow \mathcal{V}.$$

For event-based visualization, the latter mapping is extended such that it takes the event instances detected in the data into account

$$\texttt{evis} : \mathcal{D} \times \mathcal{P} \times \mathfrak{P}(\mathcal{E}) \rightarrow \mathcal{V}.$$

The formal description of event-based visualization is summarized in Definition 4.1. This formal definition is substantiated with concrete means for specifying events, detecting them, and integrating them into the visualization process later in this chapter.

Definition 4.1 (Event-based visualization) *Assumed $ET_{relevant} \subseteq ET$ describes a set of event types considered to be relevant with respect to the task at hand. Then, the creation of a visual representation $V = \texttt{evis}(D, P, E) \in \mathcal{V}$ of a dataset $D \in \mathcal{D}$, taking into account visualization parameters $P \in \mathcal{P}$ as well as the set of events $E = \texttt{detect}(ET_{relevant}, D) \subseteq \mathcal{E}$ detected from D is called event-based visualization.*

4.4. Event Specification

The event specification is the interface between the informal interests users have in their minds and the formal model of event-based visualization. The specification of event types is a prerequisite to a later detection of the users' interests in the considered data. In this section, formal means for specifying event types are introduced. To narrow the gap between the informal user and the formal means, a user-centered model of event-specification is developed, and visual interfaces for specifying events are suggested.

It must be mentioned that in several fields of Computer Science there is much ongoing research that regards event specification. The question of event specification has been broadly discussed in literature in the last decade (eg. [Gehani et al., 1992a], [Gehani et al., 1992b], [Zhang and Unger, 1996], [Chakravarty and Shahar, 1999], [Wu and Dube, 2001], or [Cheng and Gadia, 2002]). However, the approaches address research fields different from visualization; most efforts have been made on databases, simulation, and artificial intelligence. Furthermore, the approaches are very complex, and hence difficult to handle[1]. Actual implementations of the approaches – if available

[1] Recently, [Galton and Augusto, 2002] noticed that some of the formerly proposed approaches are semantically incorrect.

– are specific to their particular application field and difficult to apply to the problems addressed in this work.

From the visualization point of view, there has been no necessity to develop means for event specification, because until now interesting event types have been specified by visualization designers, who directly implement event types into the visualization software. For the approach presented in this thesis, an event specification schema is required that addresses the specific needs of the particular users that have been assigned with the task of specifying event types. It is necessary to provide a formalism that can be easily operated by using different means of interaction (e.g., simple alphanumeric input or visual event specification mechanisms).

4.4.1. Formal Means for Event Specification

To enable the detection of events in some data, event types must be described formally (i.e., in a way that the computer can evaluate them). Formal means that can be used for this purpose are introduced in this section. Since the introduced formalisms rely on relational database terminology, basic relational notions shall be briefly recapitulated (see [Maier, 1983], [Atzeni and de Antonellis, 1993], or see Section 2.2). It is assumed that a set of attributes

$$\mathcal{A} = \{A_i \mid 0 \leq i \leq n, n \in \mathbb{N}\}$$

and a set of value ranges

$$\mathcal{VR} = \{VR_i \mid 0 \leq i \leq m, m \in \mathbb{N}\}$$

are given. The union of all value ranges is called the universe

$$\mathcal{U} = \bigcup_{i=0}^{m} VR_i.$$

Each attribute of $\mathcal{A}$ is associated with a value range of $\mathcal{VR}$ by means of a mapping

$$\texttt{range} : \mathcal{A} \rightarrow \mathcal{VR}.$$

A subset $\mathcal{T} \subseteq \mathcal{A}$ is called a relation schema. A relation T on $\mathcal{T}$ is a finite set of tuples

$$T = \{\texttt{t} \mid \texttt{t} : \mathcal{T} \rightarrow \mathcal{U},\ \texttt{t}(A_i) \in \texttt{range}(A_i),\ A_i \in \mathcal{T}\}.$$

In the following it is assumed that a dataset is given as a relation T. Consequently, abstract event types that are helpful in the context of relational datasets are formally introduced and illustrated by examples.

Tuple Events and Attribute Events

Two basic event domains can be addressed in relational data structures: Event types can be defined either with respect to tuples, in which case events are called *tuple events*, or with respect to attributes, in which case the term *attribute event* is used

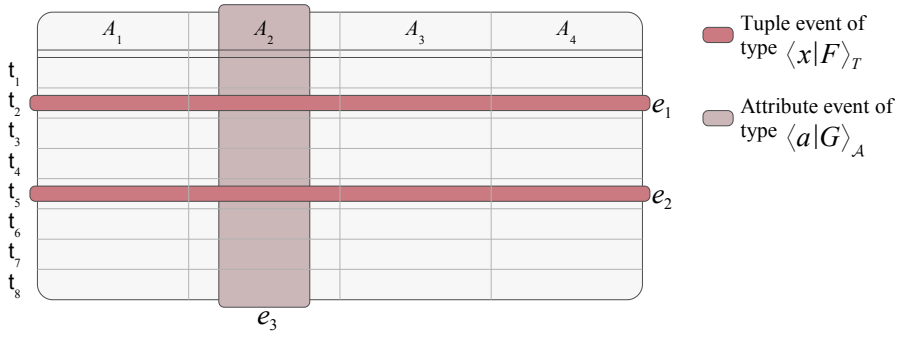


Figure 4.4.: Tuple events and attribute events.
The figure schematically depicts the scope of tuple events and attribute events in a relational dataset. In this figure $A_1, \ldots, A_4$ denote attributes and $\mathtt{t}_1, \ldots, \mathtt{t}_8$ are tuples of the dataset. e_1 and e_2 are tuple events of type $\langle x \mid F\rangle_T$ where x is a tuple variable and F is an event formula that states a condition of interest with respect to tuples. e_3 is an attribute event of type $\langle a \mid G\rangle_{\mathcal{A}}$ where a is an attribute variable and G is an event formula that states an interesting condition with respect to attributes.

(see Figure 4.4). Generally, interests can be described using conditions expressed in first order predicate logic (PL-I) formulas. Similar formulas are available in the tuple relational calculus, which has been developed to express and evaluate relational data queries (see [Atzeni and de Antonellis, 1993]). However, using the tuple calculus, it is possible to express interests only with respect to tuples, but not with respect to attributes. Furthermore, the tuple calculus in its original form neither allows functions to be applied to data values nor aggregate functions to be applied to attributes.

Therefore, a more general form of formulas is developed, which are referred to as *event formulas*. These event formulas are extensions of classic tuple calculus formulas. The alphabet of event formulas comprises the elements of the set of *tuple variables* TV, the set of *attribute variables* AV, the set of *functions* FU, the set of *predicates* PR, the set of *logical operators* LO, and the set of *quantifiers* QU. The variables are used as placeholders for entities of some event domain. They are required to describe an interest with respect to an entity without having an actual entity at hand. For event formulas, two different types of variables are distinguished: Tuple variables

$$TV = \{x, y, \ldots\}$$

are placeholders for tuples and attribute variables

$$AV = \{a, b, \ldots\}$$

are placeholders for attributes. The set

$$FU = \{\mathtt{f}_i^k \mid \mathtt{f}_i^k : \Gamma_1 \times \ldots \times \Gamma_k \to \mathcal{U}\}$$

contains functions, and the set

$$PR = \{P_i^k \mid P_i^k \subseteq (\Gamma_1 \times \ldots \times \Gamma_k)\}$$

predicates that might appear in an event formula. Here, $i = 1, 2, \ldots$ denotes a differentiation index for functions and predicates, and $k = 0, 1, 2, \ldots$ denotes the respective arity. The symbols $\Gamma_j \mid j = 0, 1, 2, \ldots, k$ are used for readability purposes and must each be replaced by either $\Gamma_j = \mathcal{A}$ or $\Gamma_j = \mathcal{U}$. As such, functions (e.g., $+$, $*$, or `avg`) and predicates (e.g., $=$ or $\leq$) are defined over the set of attributes and the universe. The set of logical operators contains at least

$$LO = \{\neg, \wedge\},$$

but can also contain derivable operators like $\vee$, $\rightarrow$, or $\leftrightarrow$. The set of quantifiers is defined as

$$QU = \{\forall_T, \exists_T, \forall_{\mathcal{A}}, \exists_{\mathcal{A}}\}.$$

Here, the differently indexed quantifiers address tuples (T) and attributes ($\mathcal{A}$).

Using variables and functions, atoms of a formula can be defined. Subsequently, these atoms are used for the recursive construction of event formulas.

Definition 4.2 (Atom) *An atom is recursively defined as follows:*

1. *If $a \in AV$, then a is an atom.*
2. *If $A_i \in \mathcal{A}$, then A_i is an atom.*
3. *If $x \in TV$ and $A_i \in \mathcal{A}$, then $x.A_i$ is an atom.*
4. *If $x \in TV$ and $a \in AV$, then $x.a$ is an atom.*
5. *If $\alpha_1, \ldots, \alpha_k$ are atoms and $f_i^k \in FU$, then $f_i^k(\alpha_1, \ldots, \alpha_k)$ is an atom.*

Definition 4.3 (Event formula) *An event formula is recursively defined as follows:*

1. *If $\alpha_1, \ldots, \alpha_k$ are atoms and $P_i^k \in PR$, then $P_i^k(\alpha_1, \ldots, \alpha_k)$ is an event formula.*
2. *If F is an event formula, then $\neg F$ is an event formula.*
3. *If F and G are event formulas, then $F \wedge G$ is an event formula.*
4. *If F is an event formula and $x \in TV$, then*
   a) *$\forall x \in T : F$ denoted by $\forall_T x : F$ and*
   b) *$\exists x \in T : F$ denoted by $\exists_T x : F$*

   are event formulas.
5. *If F is an event formula and $a \in AV$, then*
   a) *$\forall a \in \mathcal{A} : F$ denoted by $\forall_{\mathcal{A}} a : F$ and*
   b) *$\exists a \in \mathcal{A} : F$ denoted by $\exists_{\mathcal{A}} a : F$*

   are event formulas.

In event formulas, $x.A$ and $x.a$ denote access to a concrete data value. To state a condition of interest, functions, predicates, and logical operators can be used to arbitrarily combine accessed data values. Respective indices at quantifiers denote whether they quantify over tuples or attributes. Variables can occur in an event formula either free or bound. This distinction is necessary to allow the definition of valid event formulas. A tuple variable x is considered to be bound in F if F contains a sub-formula of the form $\forall_T x : G$ or $\exists_T x : G$ otherwise, x is free. Analogously, an attribute variable a is considered to be bound in F if F contains a sub-formula $\forall_{\mathcal{A}} a : G$ or $\exists_{\mathcal{A}} a : G$.

Event formulas can now be used to define tuple event types and attribute event types. For doing so, certain constraints must be imposed on the freedom of the variables appearing in event formulas.

Definition 4.4 (Tuple event type) *A tuple event type is a structure of the form $et^T = \langle x \mid F \rangle_T$ for which the following constraints hold:*

1. *F is an event formula.*
2. *x, called the target, is the only free tuple variable in F.*
3. *All other variables in F are bound.*

The set $ET_T = \{et^T \mid et^T \text{ is tuple event type}\}$ comprises all event types that obey the stated constraints. This enables the definition of the abstract tuple event type as $\widehat{T} = (T, ET_T)$.

Definition 4.5 (Attribute event type) *An attribute event type is a structure of the form $et^{\mathcal{A}} = \langle a \mid F \rangle_{\mathcal{A}}$ for which the following constraints hold:*

1. *F is an event formula.*
2. *a, called the target, is the only free attribute variable in F.*
3. *All other variables in F are bound.*

The set $ET_{\mathcal{A}} = \{et^{\mathcal{A}} \mid et^{\mathcal{A}} \text{ is attribute event type}\}$ comprises all event types that obey the stated constraints. This enables the definition of the abstract attribute event type as $\widehat{\mathcal{A}} = (\mathcal{A}, ET_{\mathcal{A}})$.

The preceding definitions describe that event types are specified by relating data values or attributes by means of predicates. Therefore, predicates are crucial for the event specification. Moreover, predicates can be logically combined to express more complex conditions. Another important aspect is that multiple tuples or attributes can be considered in a condition using the respective quantifier notation.

Tuple and attribute events occur once the condition expressed as the event formula of the event type is matched by tuples or attributes of the considered data. As such, tuple events and attribute events are a first elementary possibility to describe specific interests regarding a dataset given in relational form.

Special Predicates for Event Definition

So far, only standard predicates like $=$ or $\leq$ have been considered for relating data values; special semantics of certain value ranges have not been considered for the event specification. To allow an intuitive definition of interesting event types, it makes sense to provide expressive predicates that address the characteristics of particular value ranges. In this section, predicates are discussed that address temporal and geo-spatial data, which are particularly important in many application scenarios. It must be mentioned that it is not the goal of this section to develop complete temporal algebras or logics (e.g., [Dey et al., 1996], [Ma and Knight, 1996], [Tansel, 1997]) or novel predicates for spatial reasoning (e.g., [Randell et al., 1992]), but to demonstrate how dedicated temporal and spatial predicates can be used to ease event definition.

Especially when facing time-dependent data, it is useful to provide temporal predicates to allow users to define their interests with respect to time more directly. Temporal predicates can be differentiated into predicates that address time points and predicates that address time intervals [Hajnicz, 1996][2]. When considering time points, the temporal predicates *equals* and $before$, as well as respective inverse predicates *not-equal* and $after$ can be utilized [Hajnicz, 1996]. As such, the predicates for time points are quite similar to the usual $<, =, \neq, >$ predicates. A larger set of predicates is available for time intervals. According to Allen (see [Allen, 1991], [Allen and Ferguson, 1994]), the following predicates can be applied to intervals – $equals$, $before$, $meets$, $overlaps$, $during$, $starts$, and $finishes$. For each of these predicates, a respective inverse predicate does exist. An overview of some of the listed predicates is depicted in Figure 4.5. The reader is referred to the original work of [Allen, 1991] for details on these predicates.

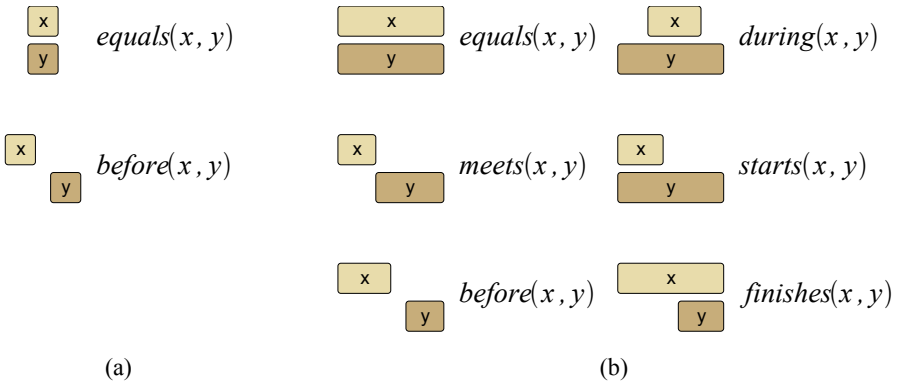


Figure 4.5.: Temporal predicates.
(a) Predicates for time points; (b) Predicates for time intervals, respective inverse predicates are possible as well.

[2]The interested reader is also referred to [Frank, 1998] for further classification criteria and different types of time.

Spatial relationships are also of interest when specifying events. A comprehensive collection of spatial predicates that can be used to relate spatial regions to each other is presented in [Randell et al., 1992]. These predicates will be briefly described to illustrate how event formulas can be enhanced with expressive spatial predicates. The predicates proposed in [Randell et al., 1992] allow statements about whether regions are equal or not. It is also possible to determine how regions are connected: Are two regions externally connected, are they internally connected, or are they disconnected? Respective inverse predicates are suggested as well. The meaning of these predicates is depicted in Figure 4.6. Detailed axiomatizations and further references on spatial predicates can be found in [Randell et al., 1992] or [Erwig and Schneider, 2002].

Using the described temporal and spatial predicates, a variety of conditions can be expressed. It becomes clear that the expressiveness of the available predicates influences the overall expressiveness of event formulas, and that adequate predicates reduce the efforts to be made by users when specifying the conditions they are interested in. This implies that in different application fields, specific predicates that meet the particular needs of that field should be made available for event specification.

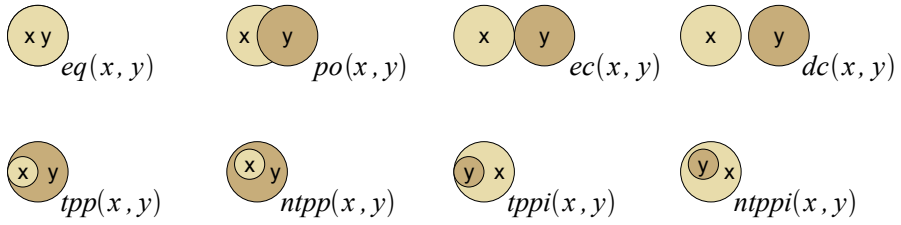


Figure 4.6.: Spatial predicates.
The figure shows a pairwise disjoint set of base predicates for spatial regions [Randell et al., 1992]. These predicates are *eq* (equals), *po* (partially overlaps), *ec* (is externally connected with), *dc* (is disconnected from), *tpp* (is tangential proper part of), *ntpp* (is non-tangential proper part of), *tppi* (inverse of *tpp*), and *ntppi* (inverse of *ntpp*).

Examples for Event Types

To demonstrate how users can specify conditions they are interested in, some examples of tuple event types and attribute event types will be given in this section. For the examples, a climate dataset T is assumed in which date, region, average temperature, precipitation, air pressure, and cloud cover have been stored on a daily basis (see Table 4.2):

$$\mathcal{T} = \mathcal{A} = (Date, Region, AvgTemp, Prec, AirPress, Cloud)$$

$$\mathcal{VR} = (DateRange, RegionRange, \mathbb{R})$$

$$\texttt{range}(A_i) = \begin{cases} DateRange & | \quad A_i = Date \\ RegionRange & | \quad A_i = Region \\ \mathbb{R} & | \quad A_i \in (AvgTemp, Prec, AirPress, Cloud) \end{cases}$$

Date	Region	AvgTemp	Prec	AirPress	Cloud
2005-01-01	HRO	2,1	12,3	987	64
2005-01-03	NVP	3,5	14,2	982	78
...	...	...	...	...	...

Table 4.2.: Climate-related sample dataset.

The set of functions FU contains common functions that can be applied to dates and real numbers; the set of predicates PR contains ordinary comparison predicates like $<$, $\leq$, or $=$, as well as the presented temporal and spatial predicates. For better readability, an infix notation is used (e.g., $a + b$ or $c \leq d$, rather than prefix $+(a, b)$ or $\leq (c, d)$). With these assumptions the following event types can be constructed.

Example 4.1 *A first, very simple event type detects a tuple with a specific value for an attribute. In particular, the event type*

$$precipitationFree = \langle x \mid x.Prec = 0 \rangle_T$$

can be used to detect tuples that describe a day with no precipitation.

Example 4.2 *Event types that describe threshold exceedances are commonly required. The type of an event that will be detected whenever the average temperature drops below 0 °C could be*

$$frost = \langle x \mid x.AvgTemp \leq 0 \rangle_T.$$

Example 4.3 *Tuple event types that relate the target tuple variable to other tuple variables can be used to state more complex conditions. For this example, the hottest day in the dataset can be described by the event type*

$$hottestDay = \langle x \mid \forall_T y : x.AvgTemp \geq y.AvgTemp \rangle_T,$$

relating the target variable (x) to all other tuples ($\forall_T y$) in the dataset.

Example 4.4 *Attribute event types are of importance if aggregate functions over attributes are considered. In this example, the set of functions FU is extended with the aggregate function $\texttt{dev} : \mathcal{A} \rightarrow \mathcal{U}$, which can be used to calculate the standard deviation of an attribute. Then, the attribute with the highest degree of change can be detected using the following event type*

$$highestChange = \langle a \mid \forall_{\mathcal{A}} b : \texttt{dev}(a) \geq \texttt{dev}(b) \rangle_{\mathcal{A}}.$$

Example 4.5 *This example shall demonstrate the use of the presented special predicates for spatial regions. In particular, the* **ec** *predicate is used to find neighbors of a region. With the event type*

$$neighborsLower = \langle x \mid \forall_T y : (y.Date \textbf{ equals } x.Date \wedge y.Region \textbf{ ec } x.Region) \rightarrow y.AvgTemp < x.AvgTemp \rangle_T$$

all regions whose neighboring regions have lower average temperatures can be detected.

Example 4.6 *A last example describes how multiple tuple variables and the $\exists_T$ quantifier can be used to detect regions with three consecutive increases in temperature.*

$$threeDayIncrease = \langle x \mid \exists_T y : (x.Date - 1 \textbf{ equals } y.Date \wedge \exists_T z : (y.Date - 1 \textbf{ equals } z.Date \wedge x.Region \textbf{ eq } y.Region \wedge x.Region \textbf{ eq } z.Region \wedge x.AvgTemp > y.AvgTemp \wedge y.AvgTemp > z.AvgTemp))\rangle_T.$$

These examples show that a variety of interesting conditions can be described using the proposed event types. However, especially by the last example, it also becomes clear that certain conditions are cumbersome to define. If users, with respect to the last example, are interested in not only three consecutive increases, but ten or more, the problem will become even worse. The respective event type could hardly be defined by inexperienced users. The reason being that tuples are not related to each other by definition, i.e., every tuple must be manually set in relation to a consecutive tuple. Another abstract event type addressing this issue will be introduced in the next section.

Sequence Event Types

From the previously presented examples it can be seen that interesting event types are sometimes difficult to specify, because multiple tuple variables have to be interrelated manually. To facilitate the event specification, it makes sense to assume a naturally given relation between tuples. A sequence of tuples is such a natural relation: It imposes an ordering of tuples, meaning that every tuple has a well-defined predecessor and successor, which is particularly useful for dynamic and/or temporal datasets. In fact, time series data that are given in relational form require the interpretation of tuples as sequence of tuples to allow for effective data analysis. Therefore, a further abstract event type – the *sequence event type* is introduced in this section. The term sequence is used to emphasize the applicability of these event types for finding events of interest in time series or other sequenced data.

The problem of how to query sequenced data is not new to database researchers. Especially in the field of temporal databases [Snodgrass, 1999], much effort has been made to answer this question. Still, most of the developed approaches are very complex, difficult to handle, and difficult to communicate to end-users. Therefore, a simple yet expressive method is required allowing sequence events to be integrated into event-based visualization. Such an elegant method has been published recently in [Sadri et al., 2004]. There, a method for expressing queries with respect to sequences is described. The proposed syntax is simple and can be easily adapted to allow the specification of events with respect to sequences of tuples.

Since sequences are not given per se in relational datasets, it is necessary to compile them using an adequate notation. This can be realized by designating both a list of identification attributes IA as well as a list of ordering attributes OA. The identifica-

tion attributes[3] are used to define for which key values separate sequences are created. The ordering attributes define according to which attributes the sequences are ordered [Sadri et al., 2004]. Table 4.3 depicts an example of how sequences can be derived from an unordered relational dataset.

Date	Region	AvgTemp	Prec	AirPress	Cloud
2005-01-01	HRO	2,1	12,3	987	64
2005-01-03	NVP	3,5	14,2	982	78
2005-01-01	DBR	0,9	10,8	987	56
2005-01-02	NVP	1,9	1,4	991	40
2005-01-01	NVP	-0,3	9,4	986	57
2005-01-03	HRO	4,3	17,3	981	81
2005-01-04	HRO	3,9	0	998	35
2005-01-02	HRO	3,2	2,9	988	45
2005-01-04	NVP	1,6	0	989	39
2005-01-03	DBR	3,2	15,2	981	75
2005-01-02	DBR	2,2	3,4	991	44
...	...	...	...	...	...

unordered →

identification attributes *IA*: Region
ordering attributes *OA*: Date

Date	Region	AvgTemp	Prec	AirPress	Cloud
2005-01-01	NVP	-0,3	9,4	986	57
2005-01-02	NVP	1,9	1,4	991	40
2005-01-03	NVP	3,5	14,2	982	78
2005-01-04	NVP	1,6	0	989	39
...	...	...	...	...	...
2005-01-01	DBR	0,9	10,8	987	56
2005-01-02	DBR	2,2	3,4	991	44
2005-01-03	DBR	3,2	15,2	981	75
...	...	...	...	...	...
2005-01-01	HRO	2,1	12,3	987	64
2005-01-02	HRO	3,2	2,9	988	45
...	...	...	...	...	...

sequenced

Table 4.3.: Sequence creation with identification and ordering attributes.
The figure shows an example of how identification and ordering attributes can be used to define sequences of tuples. For each region, the tuples are separately ordered by date.

It is now possible to extend the introduced tuple event type to support event specification with respect to sequences of tuples. One extension is made regarding the variables that are involved in the specification of an event type. Whereas for tuple event types a single tuple variable is used to express interests (e.g., $\langle x \mid \ldots \rangle_T$), for sequence event types, sequences of tuple variables are used. The additional syntactic notation required to create such sequences of tuple variables can be easily realized by comma separating multiple tuple variables (x, y, z). To allow the definition of recurring conditions with respect to a tuple variable, the notation $*y$ can be used. The $*$ denotes an arbitrary number of repetitions of the connected tuple variable (e.g., $(u, *v, *w)$) [Sadri et al., 2004]. Formally, a *tuple variable sequence* can be defined as

$$\varsigma = (\gamma_1, \ldots, \gamma_n)^{IA}_{OA}$$

where $n \in \mathbb{N}$ and each $\gamma_j \mid 1 \leq j \leq n$ is either a unique tuple variable (e.g., x) or a $*$ followed by a unique tuple variable (e.g., $*y$). By unique it is meant that a tuple variable must not appear in more than one γ_j. IA and OA denote the identification attributes and ordering attributes as introduced earlier.

A second extension that aims at easing the specification of sequence event types regards the event formulas. In [Sadri et al., 2004] it is suggested to allow access to the tuple that precedes another tuple. This is helpful, particularly for defining event types that involve the $*$ notation. The access to a tuple that precedes another tuple is denoted by the keyword previous (e.g., x.previous).

[3] In [Sadri et al., 2004] the keywords CLUSTER BY and SEQUENCE BY are used. These keywords have not been adopted due to their different meaning in the visualization community.

The set S of all possible sequences that could be created from an arbitrary number of tuples of a relational dataset, the introduced tuple variable sequences, and the previous notation are the basis for the following definition of sequence event types.

Definition 4.6 (Sequence event type) *A sequence event type is a structure of the form $et^S = \langle \varsigma \mid F \rangle_S$ for which the following constraints hold:*

1. *F is an event formula with extended tuple access (by means of previous).*
2. *ς is a tuple variable sequence.*
3. *The tuple variables constituting the tuple variable sequence ς are the only free tuple variables in F.*
4. *All other variables in F are bound.*

The set $ET_S = \{et^S \mid et^S \text{ is sequence event type}\}$ comprises all event types that obey the stated constraints. This enables the definition of the abstract sequence event type as $\widehat{S} = (S, ET_S)$.

By using sequence event types, interesting conditions that involve multiple tuples can be specified more easily. In fact, sequence event types enable the detection of interesting intervals within ordered (i.e., sequenced) relational datasets. It must be mentioned that sequence event types can only be specified with respect to datasets that contain attributes whose value ranges allow an ordering of the values. Sequence event types cannot be applied to data that contain only nominal attributes. However, this is not really a limitation, since the main application of sequence event types is to detect interesting spans in time series data, which always can be sequenced according to time. In the following, two examples will be given to illustrate the intuitiveness of sequence event types.

Example 4.7 *The event type described in Example 4.6 is redefined as a sequence event type that uses three sequenced tuple variables to detect three consecutive increases in temperature.*

$$seq3DayIncrease = \langle (x, y, z)_{Date}^{Region} \mid x.AvgTemp < y.AvgTemp \wedge y.AvgTemp < z.AvgTemp \rangle_S$$

Example 4.8 *In this example (inspired by [Sadri et al., 2004]), the use of the $*$ and the keyword previous is demonstrated. It is the goal to find five successive days with increasing temperature and an overall increase in temperature of more than 50%.*

$$seq5Day50Increase = \langle (x, *y, z)_{Date}^{Region} \mid y.AvgTemp > y.\textsf{previous}.AvgTemp \wedge 1,5 \cdot x.AvgTemp < z.\textsf{previous}.AvgTemp \wedge x.Date + 5 \textbf{ equals } z.\textsf{previous}.Date \rangle_S$$

Composite Events

The introduced tuple, attribute, and sequence event types are basic means for specifying conditions of interest with respect to relational data. What is required in the end is a method for combining arbitrary event types to composite event types. By allowing event composition, once defined event types can be reused for the specification of new and more complex events. However, the abstract tuple, attribute, and sequence event type are separate and cannot be combined with each other, because each of these abstract event types addresses a different event domain. Therefore, it is necessary to find a common event domain for composite events. The visualization (or external) time $\mathfrak{T}_e$ can be used as a common anchor for composite events.

For the specification of composite event types, all event types in ET can be combined in composite event formulas. Since composite events are always related to the current visualization time, special temporal variables are not required. The simple syntax of composite event formulas is defined as follows.

Definition 4.7 (Composite event formula) *A composite event formula is recursively defined by the following points:*

1. *Every event type $et \in ET$ is a composite event formula by definition.*
2. *If F is a composite event formula, then $\oslash F$ is a composite event formula.*
3. *If F and G are composite event formulas, then $F \cup G$ and $F \cap G$ are composite event formulas.*

In this definition, $\oslash F$ denotes a non-occurrence of F. The notion $F \cup G$ describes the occurrence of any F or G, whereas $F \cap G$ is used to denote the occurrence of both F and G. In this sense, composite event formulas have an expressive power equal to the expressive power of Boolean algebras.

Definition 4.8 (Composite event type) *A composite event type is a structure of the form $et^C = \langle F \rangle_C$ where F is a composite event formula. The set $ET_C = \{et^C \mid et^C$ is composite event type$\}$ comprises all composite event types. This enables the definition of the abstract composite event type as $\widehat{C} = (\mathfrak{T}_e, ET_C)$.*

Since composite event types may involve multiple basic event types, they can be considered a means for bringing all possible event types together. It must be kept in mind that instances of composite event types are only valid with respect to the current visualization time, i.e., after the event detection (see Figure 4.7). This is critical if the event detection and the event representation must be conducted frequently.

4.4.2. A Model for User-Centered Event Specification

In the previous sections, a variety of formal means for specifying event types have been developed. This section addresses the question of how users can specify their interest using the proposed event type formalisms. The event specification is a crucial issue in

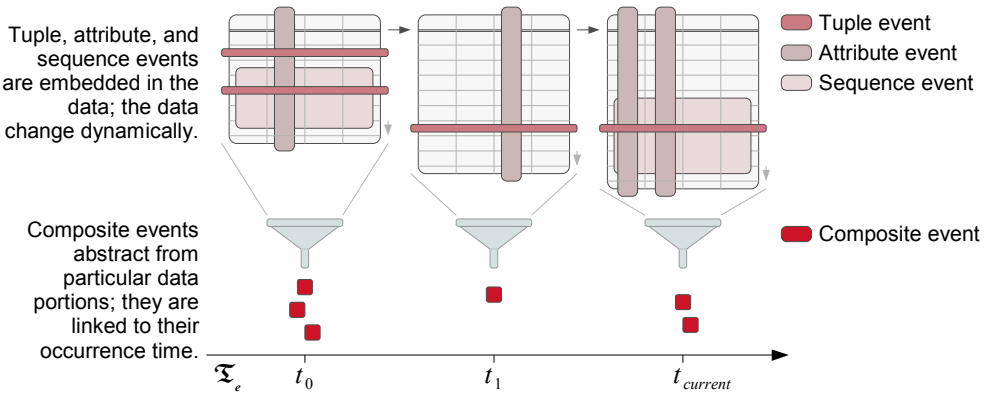


Figure 4.7.: Difference between basic and composite events.
The figure depicts tuple, attribute, and sequence events occurring in the data. In contrast to that, composite events, which may comprise arbitrary event types, occur with respect to the visualization time.

event-based visualization. If users experience difficulties in describing the events they are interested in, the whole approach becomes questionable. Therefore, much effort must be spent on supporting users in the event specification. This means that users must be provided with suitable means allowing them to intuitively specify event types, to easily alter specified event types, and to select event types currently relevant for a particular visualization session. It is also necessary to take the expertise of different users into account: Novice visualization users must be provided with tools different from those provided for expert data analysts. A three-level event specification model (see Figure 4.8) has been developed that addresses this issue. The different levels of the event specification model are:

1. Direct specification,
2. Specification by parameterization, and
3. Specification by selection.

This model uses the proposed event types and the respective event formulas as a basis. Depending on available functions and predicates, event formulas are a powerful means for describing event types. Thus, the complete functionality of these formulas should be made available to expert users. This is the case for the first level of the model, at which event formulas are specified directly. By allowing a direct specification of event formulas it is ensured that experts can specify any event type they are interested in. It must be mentioned that direct specification does not mean that event types can only be entered by typing them on the keyboard. Quite the opposite, it is important to support even expert users in the direct specification of events. This support can be achieved by applying visual approaches (methods that are particularly suitable for this purpose are described in Section 4.4.3).

On the other hand, inexperienced users might have difficulties in using event formulas directly. Therefore, it is suggested to provide predefined event type templates, which merely need to be parameterized by users. Even though event type templates use event formulas internally, they allow access only to parameters that can be easily set by users. Since the complexity of event formulas is hidden from the user, the event specification is significantly eased, and hence, the user acceptance of event-based visualization is increased. An example for an event type template is a threshold template with parameters for the attribute addressed and the particular threshold value. Users need no knowledge of the event specification formalisms, but instead can easily select an attribute of some dataset and set a concrete threshold value to specify their interest.

The third suggested level of event specification is based on simple selection. The idea is to provide a collection of predefined event types, from which users can select the most interesting ones for the task at hand. Apparently, the collection of event types to choose from has to be compiled by expert users or visualization designers. To allow an easy selection of relevant event types, it is also required that all event types provide expressive identifiers and an expressive verbal descriptions of the conditions intended to detect. This third level of event specification addresses not only novice users, but also users that are in the position of a decision maker, or any other users who seek quick access to relevant information contained in some data.

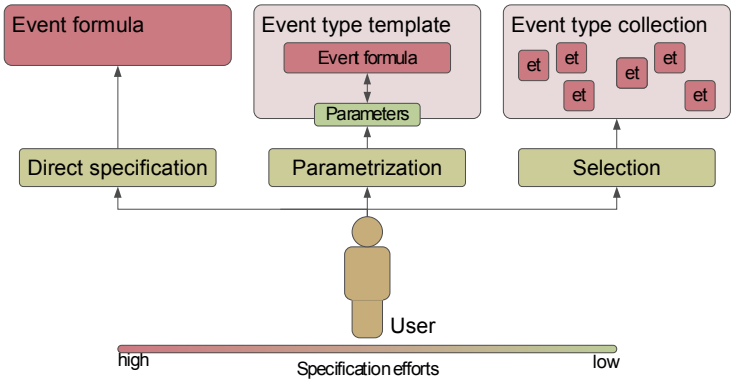


Figure 4.8.: Model for user-centered event specification.
This schema illustrates that users can specify their interests by direct definition of event formulas, by parameterization of event type templates, or by selection from an event type collection.

In Figure 4.8, the three levels of the user-centered event specification model are illustrated. The figure shows that the effort required for the specification of events is high for direct specification and low for specification by selection; the effort required for parameterizing event type templates is somewhere in between, depending on the parameters that need to be set. The introduced model of event specification ensures that event-based visualization can be operated by users with different experience. Apparently, the model implies a tradeoff between expressiveness and the level of user expertise

needed for event specification. According to [Tang et al., 2004], this is generally the case whenever formal means must be handled by human operators.

Another user-related issue that should be addressed regards the fact that different users are interested in different events occurring in different datasets. This means that whenever users visualize a certain dataset, they always have to manually specify the event types that should be considered. Especially for data analysts who have to evaluate several different datasets, this leads to additional specification effort. One possibility to alleviate this problem is to establish associations between particular users (or user groups), event types, and datasets and vice versa. Such associations can be built up easily during the event specification and whenever events are used for visualization. The established associations can be used to pre-select favorite event types automatically when a user starts an event-based visualization session or connects to a different dataset. By this procedure, event-based visualization can be handled more easily and the required event specification efforts are reduced.

4.4.3. Visual Event Specification

The previous section was dedicated to the introduction of a user-centered event specification model that describes basic possibilities of how users can specify the event types they are interested in. On the level of direct specification, the full expressiveness of event formulas is made available to users. To enable users to specify event types at this level, it is possible to simply rely on typing in event formulas using the keyboard. However, alphanumeric input is not an intuitive, and rather error-prone way of specifying event types.

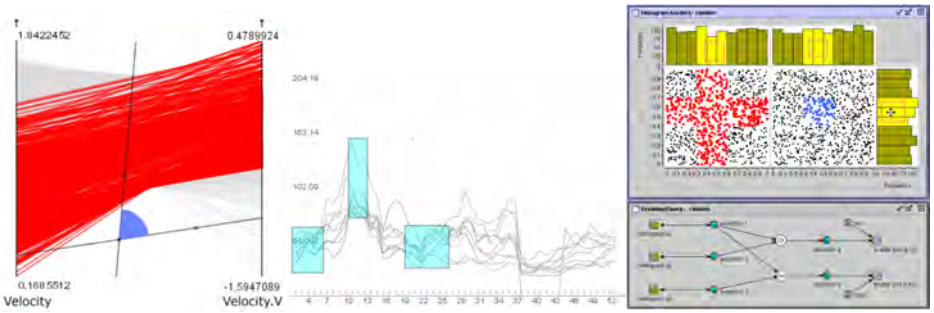
Therefore, it has been investigated how the event specification can be supported via visual methods. The goal is to use visual methods to abstract from the underlying formalism, and instead provide graphical representations and interaction techniques for specifying events of interest. Basically, two visual approaches are suitable for supporting the event specification:

- Brushing or
- Visual editors.

Brushing is usually seen as the process of interactively selecting data items from a visual representation. The original intention of brushing is to highlight brushed data items in all views of the visualization (i.e., brushing & linking [Buja et al., 1991]). In general, the idea of brushing is applied to allow users to specify data portions of interest. In [Doleisch et al., 2003], for instance, it is described how features (i.e., data portions that obey certain constraints) can be specified interactively by brushing. Another example for the applicability of brushing are Timeboxes [Hochheiser, 2003], which can be used to query time series or other linear sequences.

How brushing is actually conducted depends on the visual representation. Brushing usually means defining a two-dimensional region on the screen (e.g., by clicking and dragging the mouse to open up a rectangular region), and considering all visual primitives overlapping this region to be of interest. To determine which data items

have been brushed, it is necessary to find a reverse mapping from the visual primitives to the data. This mapping defines the semantics of the brushing operation. For simple brushing operations, the mapping step of the visualization pipeline can simply be inverted (e.g., in a Scatter Plot, where each pixel represents particular data values). However, more sophisticated brushing techniques (see Figure 4.9) consider more complex semantics. For instance, angular brushes consider slopes between attributes being adjacent in the visual representation [Hauser et al., 2002]. Timeboxes, as another example, consider properties of brushed line plots [Hochheiser, 2003]. Even compound brushing is possible [Chen, 2004], to combine multiple simple brushes via set-oriented operators.



(a) (b) (c)

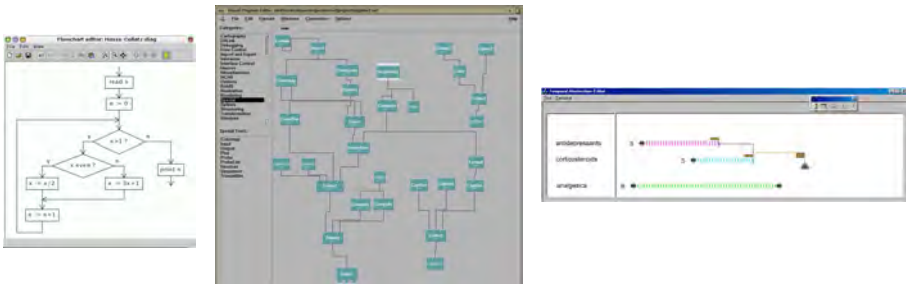
Figure 4.9.: Brushing techniques.
(a) Angular brushing [Hauser et al., 2002]; (b) Timeboxes [Hochheiser, 2003]; (c) Compound brushing [Chen, 2004].

These examples indicate that it is possible to utilize brushing to specify interesting conditions within some dataset. Depending on the concrete semantics, a variety of conditions can be specified as event types. Although brushing techniques are suitable to reduce the event specification efforts significantly, they bear a crucial disadvantage. Brushing is generally limited to the range of values already contained in the data. It is not possible to specify conditions with respect to data values that are not contained in the data (but might appear in later instances of a dataset). Therefore, further visual means are required that address this disadvantage of applying brushing for event specification.

Visual editors (or visual languages) are useful tools for this purpose. From a practical point of view, visual editors use graphical primitives and usually connections between them to allow a visual specification of some circumstances. From a theoretical point of view, visual editors allow a graphical specification of sentences of some abstract formal language [Bottoni et al., 1995]. In other words, visual editors aim at exploiting the capabilities of the human visual system for the specification of complex, formally defined structures. Visual editors are used for a variety of objectives, particularly to ease the use of query languages. In [Balkir et al., 2002], for instance, a visual query

language is described that uses an iconic interface to allow the specification of OQL-queries. Moreover, visual editors are applied for facilitating the specification of triggers for active databases [Lee et al., 2000]. Even the specification of complex temporal relationships can be supported by visual editors (see e.g., [Chittaro and Combi, 2001], [Chittaro and Combi, 2003]). An overview of approaches that make use of visual languages can be found in [Catarci et al., 1997]. Some examples of visual editors are presented in Figure 4.10. They give evidence of the broad applicability of visual editors.

Speaking in terms of a visual editor for event types, it makes sense to consider such an editor on two levels of the proposed user-centered event specification model – on the level of direct specification as well as on the level of the specification by parameterization. However, developing a visual editor is not a trivial task; a number of requirements have to be fulfilled. At first, it is necessary to take the diversity of (abstract) event types into account. Since there is not the one and only visual editor that is suitable for all different abstract event types (i.e., tuple event types, attribute event types, sequence event types, composite event types), diverse visual editors have to be developed. Moreover, there are several aspects that necessitate a common framework in which all visual event editors can be embedded. Such aspects are, for instance, project and user management, synchronization of event types with available data sources, uniform handling of visual primitives appearing in visual editors (e.g., for selecting or connecting primitives), and last but not least, extensibility to allow the integration of editors for further abstract event types. All these aspects are considered in the visual event specification framework presented in Section 5.3.



(a) (b) (c)

Figure 4.10.: Visual editors.
(a) Visual editor for flow charts [Minas, 2002]; (b) Visual editor of OpenDX [Thompson et al., 2004]; (c) Visual editor for temporal notions [Chittaro and Combi, 2003].

In conclusion, the advantage of using visual editors for event specification compared to using brushing techniques is that any event type can be specified regardless of whether the particular condition is already matched in the data or not. Of course the expressiveness of the visual editor might be a limiting factor. Another critical barrier is that of the limited screen space. This barrier has become known as the "Deutsch Limit" stated by Peter Deutsch:

> *The problem with visual programming is that you can't have more than 50 visual primitives on the screen at the same time.*

This means that visual editors are helpful only up to a certain complexity of event types to be specified. Therefore, it makes sense to incorporate the idea of event type templates (see Section 4.4.2) into visual event type editors as well. By doing so, complex sub-structures can be hidden and screen space can be saved.

From the previous discussion on visual event specification it can be inferred that visual methods are suitable means for supporting users in specifying event types. The achieved level of support depends on the particular specification method used and has much impact on the user acceptance of event-based visualization. It also became clear that different graphical means support event types of different complexity: The easier a visual editor is to handle, the less is the complexity of event types that can be specified.

4.5. Event Detection

The event detection is the second major concern of event-based visualization. It is the task of the event detection to find actual event instances of previously defined event types. Basically, finding event instances requires searching event domains for entities that conform to the interest expressed by event types.

Since there is not the one and only way to specify event types, there is, consequently, not the one and only solution to the problem of finding event instances in the data. Quite contrary, each abstract event type requires dedicated algorithms to allow an efficient detection of event instances. First, this section describes the general procedure of detecting event instances. Based on that, specific event detection methods for the previously introduced abstract event types are presented.

General Event Detection Procedure

The starting point for detecting events in a relational dataset $T \in \mathcal{D}$ is a set

$$ET_{relevant} \subseteq ET$$

of event types that a user considers to be relevant. To detect instances of these event types, it is necessary to construct a mapping

$$\texttt{detect} : \mathfrak{P}(ET) \times \mathcal{D} \rightarrow \mathfrak{P}(\mathcal{E})$$

(see Section 4.3) such that applying `detect` to $ET_{relevant}$ and T yields a set of event instances

$$E_{relevant} = \texttt{detect}(ET_{relevant}, T) \subseteq E$$

relevant in T. Internally, the mapping `detect` must consider each event type in $ET_{relevant}$ separately. To actually detect events of type $et \in \widehat{ED}$, the event domain ED must be searched for entities ed that conform to the interest expressed by the

event type. In other words, for each entity $ed \in ED$, it must be checked whether ed conforms to et. If this is the case, an event instance

$$e = (ed, et, EP)$$

is added to the set of event instances $E_{relevant}$. To find all relevant event instances, this check for conformance must be conducted for all event types in $ET_{relevant}$.

The previous description indicates that event instances are bound to entities of event domains. Additionally, an event instance preserves its event type et, and a set of event parameters EP. The set of event parameters EP is useful for storing any kind of event-related meta-data, for instance, importance information, or other calculated or user specified information. With the availability of this general set of event parameters, event instances can be easily extended to future requirements.

When detecting events from some data, it is necessary to differentiate the detection of events within:

- Static data, or
- Dynamic data.

For static data, the event detection can be performed in a preprocess, i.e., prior to the visualization. Therefore, the runtime requirements are not that important for the static case. However, for dynamic data, the event detection must be performed repeatedly, i.e., whenever changes to the data have occurred. In this case, the efficiency of the event detection becomes crucial. Especially if the data change very frequently, the event detection must be able to handle all changes in time. Certainly, this requirement can be fulfilled only up to a certain frequency of data changes. Beyond this frequency, it is impossible to detect event instances in a timely manner.

Since different event domains have different requirements regarding the event detection process, it makes sense to address different abstract event types (i.e., $\widehat{T}$, $\widehat{\mathcal{A}}$, $\widehat{S}$, and $\widehat{C}$) with separate, specifically adjusted methods for detecting event instances. In the next sections, methods for detecting tuple and attribute events, sequence events, and composite events are described in detail.

Detection of Tuple and Attribute Events

First, the detection of tuple and attribute events shall be described. Basically, two steps have to be performed for detecting such events:

- Substitution of variables appearing in an event formula and
- Evaluation of the event formula.

The substitution is a prerequisite to the evaluation of an event formula. By substitution it is meant that all variables appearing in an event formula are substituted with concrete entities of event domains: A tuple variable is substituted with a particular

tuple, and an attribute variable is substituted with an attribute. Substitutions of tuple variables and attribute variables are denoted by

$$[x/\mathtt{t}]_T \mid x \in TV,\ \mathtt{t} \in T$$

$$[a/A_i]_{\mathcal{A}} \mid a \in AV,\ A_i \in \mathcal{A}.$$

The substitution step provides concrete entities regarding which event formulas can be evaluated. The task of the evaluation step is to determine whether the entity (either a tuple or an attribute) that has been used to substitute the target variable of an event type conforms to the condition expressed by the event formula. An event formula can be evaluated to either *true* or $false$. In case of a positive evaluation, an event instance is created as described in the previous section: The matched entity, the event type, and some event parameters are used to construct the event instance.

In the following, the evaluation of event formulas will be formally described. First, it is necessary to define how atoms of event formulas are evaluated.

Definition 4.9 (Evaluation of atoms) *The evaluation* $\mathtt{eval}(\alpha)$ *of an atom* α *is defined as follows:*

1. *If* α *is of form* $\alpha = a$ *and* $[a/A_i]_{\mathcal{A}}$*, then* $\mathtt{eval}(\alpha) = A_i$.
2. *If* α *is of form* $\alpha = A_i$*, then* $\mathtt{eval}(\alpha) = A_i$.
3. *If* α *is of form* $\alpha = x.A_i$ *and* $[x/\mathtt{t}]_T$*, then* $\mathtt{eval}(\alpha) = \mathtt{t}(A_i)$.
4. *If* α *is of form* $\alpha = x.a$ *and* $[x/\mathtt{t}]_T$ *and* $[a/A_i]_{\mathcal{A}}$*, then* $\mathtt{eval}(\alpha) = \mathtt{t}(A_i)$.
5. *If* α *is of form* $\alpha = f_i^k(\alpha_1, \ldots, \alpha_k)$*, then* $\mathtt{eval}(\alpha) = f_i^k(\mathtt{eval}(\alpha_1), \ldots, \mathtt{eval}(\alpha_k))$.

This definition of the evaluation of atoms enables the definition of how event formulas are evaluated to either *true* or $false$.

Definition 4.10 (Evaluation of event formulas) *The evaluation* $\mathtt{eval}(F)$ *of an event formula* F *is defined as follows:*

1. *If* F *is of form* $F = P_i^k(\alpha_1, \ldots, \alpha_k)$*, then*

$$\mathtt{eval}(F) = \begin{cases} true & \mid \text{ If } (\mathtt{eval}(\alpha_1), \ldots, \mathtt{eval}(\alpha_k)) \in P_i^k \\ false & \mid \text{ Otherwise} \end{cases}$$

2. *If* F *is of form* $F = \neg G$*, then*

$$\mathtt{eval}(F) = \begin{cases} true & \mid \text{ If } \mathtt{eval}(G) = false \\ false & \mid \text{ Otherwise} \end{cases}$$

3. *If* F *is of form* $F = G \wedge H$*, then*

$$\mathtt{eval}(F) = \begin{cases} true & \mid \text{ If } \mathtt{eval}(G) = true \text{ and } \mathtt{eval}(H) = true \\ false & \mid \text{ Otherwise} \end{cases}$$

4. *If F is of form:*

   a) $F = \forall_T x : G$, *then*

$$\texttt{eval}(F) = \begin{cases} true & | & \text{If for all } \texttt{t} \in T \text{ holds that } [x/\texttt{t}]_T \\ & & \text{implies } \texttt{eval}(G) = true \\ false & | & \text{Otherwise} \end{cases}$$

   b) $F = \exists_T x : G$, *then*

$$\texttt{eval}(F) = \begin{cases} true & | & \text{If it exists a } \texttt{t} \in T \text{ such that } [x/\texttt{t}]_T \\ & & \text{implies } \texttt{eval}(G) = true \\ false & | & \text{Otherwise} \end{cases}$$

5. *If F is of form:*

   a) $F = \forall_{\mathcal{A}} a : G$, *then*

$$\texttt{eval}(F) = \begin{cases} true & | & \text{If for all } A_i \in \mathcal{A} \text{ holds that } [a/A_i]_{\mathcal{A}} \\ & & \text{implies } \texttt{eval}(G) = true \\ false & | & \text{Otherwise} \end{cases}$$

   b) $F = \exists_{\mathcal{A}} a : G$, *then*

$$\texttt{eval}(F) = \begin{cases} true & | & \text{If it exists an } A_i \in \mathcal{A} \text{ such that } [a/A_i]_{\mathcal{A}} \\ & & \text{implies } \texttt{eval}(G) = true \\ false & | & \text{Otherwise} \end{cases}$$

Having defined the evaluation of event formulas, it is now possible to introduce the definition of tuple and attribute event instances.

Definition 4.11 (Tuple event instance) *Assumed $\langle x \mid F \rangle_T \in \widehat{T}$ is a tuple event type, then $e = (\texttt{t}, \langle x \mid F \rangle_T, EP)$ is a tuple event instance if and only if $[x/\texttt{t}]_T$ implies $\texttt{eval}(F) = true$. EP denotes event parameters.*

Definition 4.12 (Attribute event instance) *Assumed $\langle a \mid F \rangle_{\mathcal{A}} \in \widehat{\mathcal{A}}$ is an attribute event type, then $e = (A_i, \langle a \mid F \rangle_{\mathcal{A}}, EP)$ is an attribute event instance, if and only if $[a/A_i]_{\mathcal{A}}$ implies $\texttt{eval}(F) = true$. EP denotes event parameters.*

The recursive definition of the evaluation step implies that an algorithm is required that recursively evaluates all sub formulas appearing in an event formula. This means that the algorithm needs to subdivide an event formula into its sub formulas until only elementary predicate checks have to be conducted. When leaving a level of recursion, the algorithm has to combine the partial evaluation results of the sub formulas correctly and propagate them back to determine the final result of the evaluation. A key point in this procedure is the evaluation of atoms, which enables access to actual tuple values (e.g., $x.AvgTemp$ could evaluate under $[x/\texttt{t}]_T$ to $\texttt{t}(AvgTemp) = 23,3°C$). The

elementary evaluation of predicates is also of importance (e.g., $\texttt{eval}(23, 3°C > 20°C) = \mathit{true}$).

Although the previously described evaluation procedure seems to be simple and efficient, it is actually not. The reason for this is that the substitution of the target variable must be performed for each entity of the event domain. Furthermore, if an event formula comprises multiple quantifier-bound variables, it is also necessary to iterate entity sets for the substitution of these variables. Apparently, this means that the runtime of the event detection depends on the number of variables used and the cardinality of the respective entity sets.

To perform substitutions and evaluations more efficiently, it is suggested to exploit the tight relationship between the introduced event formulas and the relational tuple calculus, and to take advantage of the enormous capabilities of modern relational database management systems (RDBMS)[4]. For doing so, a mapping must be found that transforms event formulas to SQL queries in such a way that after executing the queries on a database server, the query results contain all entities conforming to the interests stated by the event formulas. Such a mapping can be easily implemented by adopting the methods described in [Gogolla, 1990]. Because standard SQL has originally been designed for querying tuples, but not attributes, it is not possible to speed up the detection of attribute events by using SQL queries. It must also be noted that the power of RDBMS can only be utilized if the predicates appearing in event formulas are compatible with the predicates available in the RDBMS. Unfortunately, common RDBMS usually do not support all predicates presented in Section 4.4.1. Nevertheless, using RDBMS can lead to a significant increase in the efficiency of detecting larger numbers of tuple event types.

Detection of Sequence Events

For the detection of sequence events, a method is required that is capable of finding intervals of tuples that conform to the conditions expressed in an extended event formula. The problem of finding all positions of a substring (i.e., a pattern) in a sequence of characters is well-known in Computer Science. This *string matching problem* is very similar to finding intervals in tuple sequences. Since the string matching problem is equivalent to the problem of deciding whether a word belongs to a regular language, it seems apparent to make use of regular expressions for the detection of sequence events. The KPM algorithm [Knuth et al., 1977] is known to be able to solve such problems in linear time. However, sequence events, which have been described here as an adaptation of the sequence queries [Sadri et al., 2004], need a more sophisticated algorithm for their detection. The required algorithm must support predicates like those described in Section 4.4 (KPM considers only =) as well as repetitive tuple variables (KPM assumes patterns with a fixed number of elements).

An algorithm that meets these requirements is the *Optimized Pattern Search (OPS)* algorithm introduced in [Sadri et al., 2004]. The OPS algorithm is an extension of the KPM algorithm. Like KPM, OPS aims at minimizing the number of comparisons

[4]RDBMS are able to analyze and to optimize query executions to achieve better efficiency.

required to determine whether or not a pattern is contained in a sequence by exploiting logical relations among pairs of pattern elements. The logical relations, which can be derived from the used predicates, are stored in a positive and a negative precondition matrix [Sadri et al., 2004]. The precondition matrices are the basis for computing information on how to process the pattern in the search space. This information is stored in arrays called *shift* and *next*. Whereas the *shift* array describes how to advance the pattern in the input, the *next* array describes at which element of the pattern the evaluation has to be resumed after a shift. Thanks to the precomputed arrays, the OPS algorithm enables the efficient search for patterns in arbitrary input sequences. The same algorithm can be used to detect sequence events in ordered (i.e., sequenced) relational data structures. Especially when facing large time series data, the application of the OPS algorithm is indispensable. For further details on the implementation and/or the evaluation of the OPS algorithm, the interested reader is referred to the original publication [Sadri et al., 2004] or Sadri's thesis [Sadri, 2001].

By applying the OPS algorithm it is possible to determine if a sequence of tuples contains none, one, or more intervals that conform to the condition stated in the extended event formula of some sequence event type. For each interval found, a sequence event instance is created.

Definition 4.13 (Sequence event instance) *Assumed* $\langle \varsigma \mid F \rangle_S \in \widehat{S}$ *is a sequence event type, then* $e = ((\mathtt{t}_1, \ldots, \mathtt{t}_m), \langle \varsigma \mid F \rangle_S, EP)$ *is a sequence event instance, if and only if* $(\mathtt{t}_1, \ldots, \mathtt{t}_m)$ *conforms to the conditions stated in* F *(checkable via the OPM algorithm).* EP *denotes event parameters.*

Detection of Composite Events

In the previous section, methods for detecting instances of tuple, attribute, and sequence event types have been described. A further method is required for the detection of composite events. The difference of detecting composite events compared to the previously described methods is that composite events will not be linked to some data portion, but will be anchored in time. In other words, an instance of a composite event is valid only with respect to the current event detection.

Since composite event types can be composed of arbitrary event types, also including composite event types, their detection is implemented as a combination of different event detection results. Since the abstract composite event type has been introduced using simple set-theoretic notions, the detection of composite events is straightforward. An event instance of some composite event type can be detected in two steps:

1. Detection of those event types the composite event type is composed of and
2. Application of set theory to determine whether the composite event itself has occurred.

Because composite event types may comprise non-composite, but also further composite event types, the first step must be performed recursively until it is determined for all event types participating in a composite event type whether event instances

occurred or not. The result of the first step is a set E_t of event instances detected at point $t \in \mathfrak{T}_e$ in the visualization (or external) time $\mathfrak{T}_e$. In the second step, the actual composite event formula is evaluated. For doing so, the set E_t is used and the following formal rules are applied.

Definition 4.14 (Evaluation of composite event formulas) *Under the assumption that the set E_t comprises all instances of those event types a composite event formula F is composed of, the evaluation $\texttt{eval}(F)$ of the composite event formula F is defined as follows:*

1. *If F is of form $F = et$, i.e., F is simply an event type $et \in ET$, then*

$$\texttt{eval}(F) = \begin{cases} true & | \quad \text{If } E_t \text{ contains at least one event instance of type } et \\ false & | \quad \text{Otherwise} \end{cases}$$

2. *If F is of form $F = \oslash G$, then*

$$\texttt{eval}(F) = \begin{cases} true & | \quad \text{If } \texttt{eval}(G) = false \\ false & | \quad \text{Otherwise} \end{cases}$$

3. *If F is of form:*

   a) *$F = G \cup H$, then*

$$\texttt{eval}(F) = \begin{cases} true & | \quad \text{If any } \texttt{eval}(G) = true \text{ or } \texttt{eval}(H) = true \\ false & | \quad \text{Otherwise} \end{cases}$$

   b) *$F = G \cap H$, then*

$$\texttt{eval}(F) = \begin{cases} true & | \quad \text{If both } \texttt{eval}(G) = true \text{ and } \texttt{eval}(H) = true \\ false & | \quad \text{Otherwise} \end{cases}$$

The simplicity of the previously described rules allows an efficient detection of composite event instances.

Definition 4.15 (Composite event instance) *Assumed $\langle F \rangle_C \in \widehat{C}$ is a composite event type, then $e = (t, \langle F \rangle_C, EP)$ is a composite event instance if and only if $t \in \mathfrak{T}_e$ is the current visualization time, and $\texttt{eval}(F) = true$ with respect to the current event instances collected in the set E_t. EP denotes event parameters.*

Increasing Detection Efficiency

Even though the previously described methods are suitable for finding event instances of all introduced abstract event types (i.e., $\widehat{T}$, $\widehat{\mathcal{A}}$, $\widehat{S}$, and $\widehat{C}$), there is still the need for increasing the event detection efficiency. Experiences with real-world data have shown that even the most efficient query mechanisms (e.g., the OPM algorithm) reach their limits if a certain number of tuples is exceeded. Therefore, two basic concepts are suggested for increasing the event detection efficiency.

Incremental event detection The first concept is inspired by research in the field of active databases (see [Paton, 1999] or Section 2.3 on page 14), more precisely, research on the problem of how to evaluate conditions of ECA rules efficiently. A possible solution to this problem is to evaluate or monitor conditions of ECA rules incrementally (see [Baralis and Widom, 1995] and [Risch and Sköld, 1999]). In other words, the conditions are not evaluated with respect to the whole dataset, but with respect to a differential dataset that contains only the changes that have occurred since the last evaluation of the conditions. Without having to go into too much detail, it is obvious that the differential dataset will generally be much smaller than the original dataset; the evaluation can be conducted more efficiently.

The same idea can be applied to the detection of events in dynamic datasets. This will be illustrated in a short example. It is assumed that a dynamic temporal dataset is given, i.e., a dataset for which no changes are allowed except for the insertion of new tuples. When considering the threshold event from Example 4.2 on page 56, which is detected if a tuple value exceeds a certain threshold, it becomes clear that the event formula needs to be evaluated only for newly added tuples, rather than for the whole dataset. As a matter of fact, the event detection can benefit from the incremental evaluation methods developed in the field of active databases. However, the approaches are very complex and require detailed knowledge about condition checking in active databases and relational theory. Since the approaches known in literature are limited to the evaluation of conditions with respect to tuples, only the detection of tuple events can be improved. To the knowledge of the author, there exists no approach that is general enough to be applicable for the detection of attribute, sequence, or composite events. Since the development of such an approach is beyond the scope of this thesis, the idea of incremental event detection for dynamic data (based on previous work from the field of active databases) should be pursued in future work.

User-centered event detection To increase the event detection efficiency, it also makes sense to take into account how users value occurred events. Users are commonly interested in only the "newest" events found in the data. This becomes clear when imagining a physician who uses event-based visualization to monitor the current health situation (see also Section 4.2). Usually, the physician is interested in events that have occurred on the current day, maybe also all events that occurred in the last week, but definitely not in events that occurred three months ago. The fact that users are mostly interested in current events allows a reduction of the search space for the event detection to only "recent" data. Yet, the reduction should not be fixed, but must be freely adjustable, so that in cases where users need to take a look into the past they can easily access the required information (e.g., to answer the question what events occurred during the influenza season last year). How much performance can be gained by reducing the search space of the event detection in the suggested way depends heavily on the application context.

To enable user-centered event detection, the visualization system must preserve the state of the last event detection such that it can be determined where in the data the search for possibly newly occurred events has to start. Furthermore, an analysis

of the considered event types is necessary to determine if particular event types need an earlier starting point (i.e., a greater search space) to be detected. It must also be mentioned that for some event types the search space cannot be reduced. The reason for this is that the introduced event type formalisms allow the construction of event types that require access to the whole dataset to detect event instances (e.g., some sequence event types involving the $*$ notation). However, a theoretical analysis of this issue is beyond the scope of this thesis, but should be considered for future work in cooperation with database specialists.

Summary on Event Detection

From the previous discussions it becomes clear that finding event instances is not a trivial task. To establish a basis for the event detection, basic formal methods have been described. These methods are adequate for illustrating the benefits of event-based visualization. Nevertheless, in real application scenarios, it is indispensable to consider methods for increasing the detection efficiency. Two such concepts have been described – incremental event detection as well as user-centered event detection. Because both concepts bear great potential, they should be subject to more theoretical investigations in the future.

4.6. Event Representation

The last of the three major issues of event based visualization is the event representation – and it is by far the most important one for achieving the goals targeted in this thesis. Now that event instances can be detected from the data, the question at hand is how to visually represent these events to yield benefit for the data analysis. In the following, possible answers to this question are given.

4.6.1. Requirements and Fundamental Considerations

At first, it is necessary to state requirements that should be achieved by integrating the detected event instances into the visualization. Basically, there are three differently important requirements that any event-based visualization should strive to achieve:

1. Communication of the fact that something interesting has occurred.
2. Emphasis on event instances in the visual representation.
3. Conveyance of the types of event instances.

The first and most vital requirement demands that under any circumstances users must be made aware of the fact that something interesting has happened. The second requirement demands that event instances are emphasized in the visual representation. This is also a crucial requirement, since it ensures that the importance of events is reflected in the visualization. Achieving that the emphasis really draws the attention of users to the represented events is more important than the question of how the

emphasis is actually realized (e.g., by highlighting). The third requirement aims at additionally conveying event types. In other words, it should be possible for users to differentiate visualized event instances by the event type. In the light of arbitrarily definable event types, this requirement is challenging to fulfill.

Interestingly, a relation exists between the stated requirements: Each requirement can be seen as an extension of its predecessor. The second requirement can be seen as an extension of the first one in the sense that by communicating the event fact, users are enabled to perceive that something interesting has occurred, whereas by fulfilling the second requirement, users are additionally enabled to realize the concrete entities for which events have occurred, i.e., where in an event domain something interesting has been detected. In turn, the third requirement extends the second one by further enabling users to conceive the event types, i.e., what has happened (e.g., a value exceeded a threshold or a sequence of days with increasing amounts of precipitation). It becomes clear that the expressiveness of event representations is increased step by step with respect to the stated requirements. Accordingly, the challenges to fulfill the requirements grow significantly with increased expressiveness.

To achieve the established requirements, two issues must be taken care of prior to finding actual visual representations of events. The first of these issues regards the question of how events can possibly occur. The second issue concerns the interpretation of event instances, i.e., the question what could be the meaning of event instances in particular application contexts. Both issues are discussed in the following.

Properties of the event occurrence To find adequate visual event representations, it is necessary to understand how events can occur. A first possibility to classify the event occurrence is to make a distinction between event instances that:

- Might occur,
- Always occur, or
- Never occur.

To which category an event belongs can be determined by analyzing the event formulas of event types. Usually, it is the case that event instances are in the first category, i.e., event instances might occur (e.g., a value exceeds a threshold). In fact, such events are the most interesting ones from the visualization point of view. It is also possible that events do always occur. An example for an event of this category is a maximum event that detects the data tuple with the maximum data value. Apparently, this is an event type for which an event instance will always be detected, since there will always be a maximum in the data. Although it is predetermined that the maximum event will always occur, it is not known in advance for which data tuple it will occur. This aspect should be considered by the event representation. The category of never occurring events serves more or less as an exclusion criterion: If it can be predetermined that events of a particular type are impossible to occur in a visualized dataset (e.g., an event type addresses some attributes not contained in the relation schema of the dataset, or

the stated condition can, from a theoretical point, never be positively evaluated), they need not to be considered in the event representation.

Another useful criterion regarding which events can be classified concerns their reoccurrence. This criterion is only applicable for dynamic data, i.e., data that change over time. In the context of dynamic data it is possible to differentiate between event instances that:

- Occur, disappear, and then might reoccur at some time,
- Occur once and then reoccur forever, or
- Occur once and then never occur again.

An analysis of this characteristic is more difficult, since it is mainly determined by the semantics of the considered dataset. In non-temporal datasets (i.e., datasets that do not maintain a history of data values, see Section 3.1) it is common that a tuple event is detected, disappears with the next event detection, and reoccurs for later event detections. Usually, such events are relevant in monitoring scenarios where critical data values (e.g., high pressure) necessitate certain physical actions to be taken by the users (e.g., open a valve). In contrast to that, for dynamic temporal datasets (i.e., datasets that maintain a history, see Section 3.1), a once detected tuple event will always be found in subsequent event detections, i.e., the event will occur and then reoccur forever. The reason for this is that once a tuple is contained in a dynamic temporal dataset the tuple will neither be changed nor be deleted. Since the number of always reoccurring events can be large, they must be considered with care in the event representation, otherwise cluttering of the visual representation could be a consequence. One circumstance that can be exploited to limit the number of events to be represented regards the fact that always reoccurring events do yield only little information to the user. This enables considering events farther in the past with less importance in the visualization (e.g., by fading their visual representation). For the third category of events, i.e., events for which it is known that after a first event occurrence no event of the same type will ever reoccur again, it is even possible to neglect the event types after the first event occurrence. Usually, this is only possible if it is predetermined in the application context that the event occurs only once (e.g., only one runner in a race will be first place).

Event	***Might (re)occur***	***Always (re)occurs***	***Never (re)occurs***
Occurrence	highly interesting	interesting	negligible
Reoccurrence	highly interesting	interesting	negligible after first occurrence

Table 4.4.: Occurrence and reoccurrence of events.

The described categorization of occurrence and reoccurrence properties of events is summarized in Table 4.4. Additionally, the table illustrates which events are the most interesting ones to consider in the event representation. Bearing the previously described characteristics in mind facilitates finding suitable event representations.

Interpretation of event instances A second necessary task is to find meaningful interpretations of possible event instances. This interpretation is a necessity to enable the development of expressive visual means for representing events. Since event instances are always linked to entities of event domains, a possible way to interpret events is to consider the semantics of the event domains. In the case of this thesis, the event domains are strongly related to the relational data model. Accordingly, the following interpretations of event instances are useful:

Attribute events An attribute event occurs if an interesting condition holds for a particular attribute of a relation schema. Consequently, an attribute event can be interpreted as an attribute that is specifically of interest. This interpretation implies that attribute events abstract from concrete data values and focus more on the dimensions of the data space. This is an important fact to be considered in the event representation.

Tuple events If a tuple event occurs, this is an indication that an interesting tuple, i.e., an interesting observation of data values, is contained in the data. This allows two alternative interpretations of a tuple event. On the one hand, it is possible to consider a tuple event as a loose collection of interesting data values (one value for each attribute). On the other hand, a tuple event can be interpreted as an interesting point in the space that is spanned by the attributes of the considered dataset.

Sequence events Since sequence events detect intervals within ordered tuple sequences, they are more general than tuple events. Hence, in addition to the interpretation as loose collection of interesting data values or collection of interesting points in the space that is spanned by the attributes of a dataset, sequence events can also be interpreted as a collection of interesting intervals of the attributes' value ranges.

These principle interpretations are the starting point for finding expressive means for representing events. A schematic illustration of the suggested interpretations is presented in Figure 4.11. From Figure 4.11 it becomes clear that attribute, tuple, and sequence events show an increasing number of possible interpretations; the number of possibly expressive event representations increases in the same order.

What has not been discussed so far is an interpretation of composite events. These events are special in the sense that they do not have a link to some portion of a dataset: Composite events abstract from the data and instead are anchored in time. Therefore, composite events can merely be interpreted as time points of interest during a visualization session. This interpretation of composite events requires specific visual event representations.

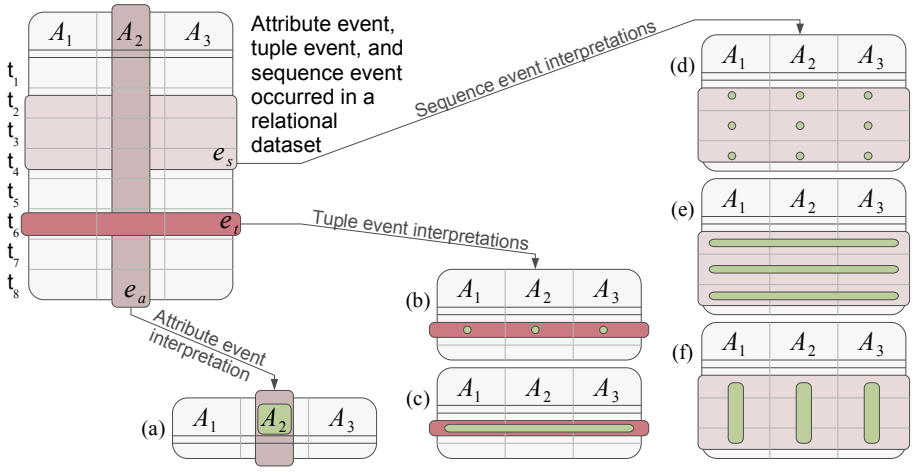


Figure 4.11.: Interpretation of events.
(a) Attribute of interest; (b)+(d) Collection of interesting data values; (c) Interesting point in the space that is spanned by the attributes; (e) Collection of interesting points in the space that is spanned by the attributes; (f) Interesting intervals of attribute values.

It is worth mentioning that it is possible to interpret any event instance (regardless of the particular event type) only as an "interesting fact". In this case, the interpretation also abstracts completely from the underlying dataset and considers only the fact that something interesting occurred in the data at some point during the visualization. Apparently, the second requirement of event representation cannot be achieved when applying such an interpretation.

For an expressive event representation, it is equally important to interpret the semantic characteristics of events. These characteristics are usually encoded by the person who maps certain interest (the semantics) to event types by using the formal means of event specification. The other way around, that is, an automatic mapping from the event formula to the semantics of an event, can hardly be realized. Therefore, event instances should be associated with certain parameters describing the event semantics. As indicated earlier, the importance of events is a useful semantic characteristic that enables an appropriately balanced representation of different events. Another interesting semantic characteristic is the scope of events. Events can have a global scope, in which case the events are detected with respect to the whole dataset. Events can also have a local scope, meaning that these events are valid only with respect to a certain part of a dataset. This distinction becomes clear when imagining a maximum event that detects maximum data values. On the one hand, the event could detect the global maximum; on the other hand, it is also possible that the event formula imposes certain constraints that restrains the maximum detection to particular parts of the data such that only local maxima are detected. Apparently, global and local events should be weighted differently in the event representation. The distinction between global and

local events is particularly important when considering spatio-temporal data, i.e., data that are related to time and space. This aspect will be explained in the following discourse on spatio-temporal events.

Discourse on spatio-temporal events In the case of spatio-temporal data, users are interested in events that are related to time and to space. Using the described temporal and spatial predicates (see Section 4.4.1), it is possible to define a variety of event types with respect to time and space. Two questions arise regarding the event representation in space and time. Firstly, are events global or local with respect to time, and secondly, are events global or local with respect to space? Depending on the particular answers to these questions, events should be represented differently. This becomes clear when looking at Table 4.5. In this table, the expected semantics of spatio-temporal events is depicted. If events are local with respect to both time and space, they detect locally interesting situations (e.g., "Is the number of influenza cases greater than a certain threshold for a particular location and a particular week?"). In the case that events are global with respect to time or space, more interesting events that are capable of detecting certain temporal or spatial patterns in the respective domain are possible (e.g., a series of value increases over a certain period of time for a particular area, or an area for which all neighboring areas have lower data values for a particular month). The complexity of events for spatio-temporal data rises if time and space are both considered globally. An event like "The maximum number of influenza cases increases significantly from one week to the next and at the same time the new maximum occurs for a different region" is an example for this event category. As already mentioned, such complex events are usually not defined by common users, but by experienced data analysts or visualization designers. The given examples indicate that local events describe rather coarse and simple interests, whereas global events tend to be subtle and difficult to describe. And, arguably, it is the subtle things that could, if represented adequately, result in an increase in expressiveness of visualization. Therefore, the considerations made in this section should be taken into account when visualizing events detected in spatio-temporal data.

		Time	
		local	global
Space	local	Simple events	Temporal pattern events
	global	Spatial pattern events	Complex spatio-temporal pattern events

Table 4.5.: Different scopes of spatio-temporal events.
Events with different scopes aid the detection of differently complex phenomena in spatio-temporal data.

Implicit and explicit event representation With the previously described interpretations and semantic characteristics in mind, it is possible to investigate expressive means for representing events visually. Basically, events can be represented in two alternative ways:

- Implicit event representation and
- Explicit event representation.

Since this distinction has already been introduced as classification criterion for event-based visualization techniques (see Section 3.1), only a brief recapitulation of implicit and explicit event representation will be given here. Irrespective of particular visualization techniques, implicit event representation concerns the automatic adjustment of the parameters of the visualization transformation to create visually perceivable event representations within the original visual representation. In this sense, event representations are merged into the visualization of the data. In contrast to that, explicit event representation concerns the visualization of solely event instances. In other words, the visual representation of events is separated from the visual representation of the data.

In the following sections, concepts for implicit and explicit event representation are described and concrete event representation methods are introduced. At first, two simple examples shall be given to illustrate implicit and explicit event representation and to clarify the difference between both approaches.

Implicit event representation in a time diagram The goal of this example is to integrate events into simple, yet powerful time diagrams. Due to their effectiveness in conveying trends and outliers, time diagrams are commonly used to represent time-dependent attributes. For this example, it is assumed that a data analyst is interested in maxima in stock data. However, due to scaling issues, it is sometimes difficult to confidently detect the maximum in a time diagram (see Figure 4.12 (a)). Therefore, the analyst specified the global maximum as an event type. Since the respective event instance, which will always be detected, is linked to the maximum stock value, it is possible to automatically highlight the maximum value as illustrated in Figure 4.12 (b). In this example, the width and color of the line segments that encode the maximum value are adjusted to achieve the highlighting. Markers have also been incorporated into the time diagram to support the detection of the highest stock value. By the described adjustments, the maximum event is implicitly represented in the time diagram. In an event-based visualization framework, the data analyst could use any technique available, each of which would provide an automatic highlighting of the maximum stock value.

Explicit event representation as Scatter Plot In this example it is assumed that several tuple events of different types have been detected in a relational dataset. Since tuple events can be interpreted as a collection of interesting data values, it is possible to represent the detected events in a Scatter Plot (see Figure 4.13). Events are visualized as dots, which are positioned with respect to two perpendicular axes that represent the value ranges of two attributes. Different colors are

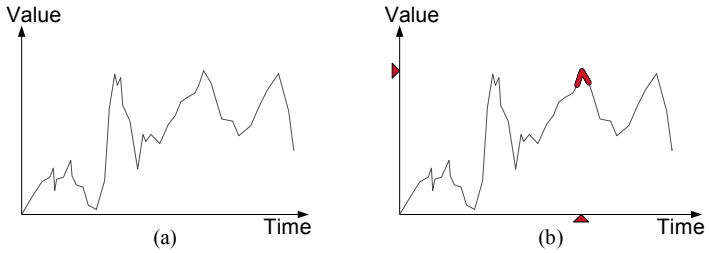


Figure 4.12.: Example of implicit event representation in a time diagram.
It is assumed that a data analyst is interested in the maximum. (a) In this diagram it is hardly possible to see which of the peaks represents the maximum. (b) By incorporating events (a maximum event in this case), the interesting parts in the time diagram can be highlighted automatically. The time diagram emphasizes the aspect the data analyst is interested in.

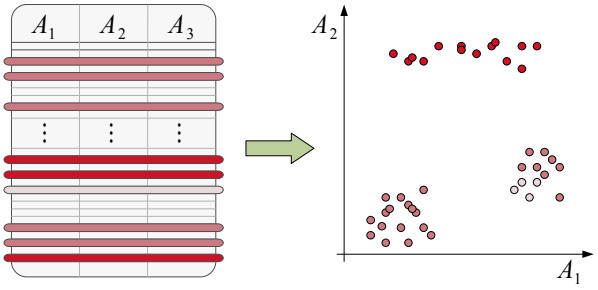


Figure 4.13.: Example of explicit event representation using a Scatter Plot.
Several tuple events have been detected in a dataset (left). The detected events are explicitly represented in a Scatter Plot (right). This representation is useful for revealing clusters of events.

used to encode different event types. By focusing on events (through representing them explicitly), relations among user-relevant aspects can be revealed and deeper, more abstract, insights into the visualized data can be gained. Although in this example only a simple Scatter Plot of events is used, it is also possible to represent events in Scatter Plot matrices[5]. This would enable the detection of correlations of events regarding all attributes of a dataset.

4.6.2. Implicit Event Representation

Implicit event representation is suggested as a means for incorporating the developed event-based concepts into known visualization techniques. By implicit event representation, the large number of available visualization approaches can be enhanced with event-based methodology without the need to develop entirely new visualization tech-

[5]Scatter Plot Matrices for visualizing multivariate data were introduced in [Cleveland, 1993].

niques. In the following, the basic procedure of incorporating events into a visualization is introduced. Based on that, concrete suggestions are made for considering event representations at different steps of the visualization pipeline.

Principles of Implicit Event Representation

A requirement for incorporating events successfully into known visualization techniques is to find an interface between visualization and events. The parameters of the visualization transformation have been identified as suitable for this purpose. The goal is to adjust visualization parameters automatically to create an implicit representation of the detected events. This implies that the addressed visualization techniques must implement the parameterizable visualization transformation (see Section 2.1), and additionally, must provide adequate and freely adjustable parameters. Only if these requirements are fulfilled by a visualization technique, is it possible to merge data and event representations. Apparently, the available parameters and their degrees of freedom determine the achievable expressiveness of implicit event representation.

If a visualization technique provides adequate visualization parameters, implicit event representations can be implemented as either:

- Instantaneous or
- Gradual

parameter adjustments. Instantaneous changes of visualization parameters are referred to as *actions*; they have immediate effect on the visualization. A simple example for an action is changing a "color" parameter from red to blue.

Definition 4.16 (Action) *An action describes an instantaneous change of visualization parameters caused by the occurrence of some event. An action can be defined as a mapping*

$$\texttt{act} : \mathcal{P} \times E \rightarrow \mathcal{P}$$

where $\mathcal{P}$ denotes the set of visualization parameters and E denotes the set of possible event instances.

In contrast to actions, *processes* cause gradual changes of visualization parameters. Processes consume time (i.e., visualization or external time, see Section 3.1) during the adjustment of parameters. An example for a process is the movement of the focal point in a focus+context visualization toward an interesting data portion that is associated with an event.

Definition 4.17 (Process) *A process describes a gradual change of visualization parameters over time caused by the occurrence of some event. A process can be defined as a mapping*

$$\texttt{proc} : \mathcal{P} \times E \times \mathfrak{T}_e \rightarrow \mathcal{P}$$

where $\mathcal{P}$ denotes the set of visualization parameters, E denotes the set of possible event instances, and $\mathfrak{T}_e$ denotes the external (or visualization) time.

This distinction between instantaneous (actions) and gradual (processes) parameter changes raises the question which possibility to use in particular situations. In other words, does it make sense to switch a particular parameter immediately to achieve an emphasis on events, or is it better to apply a gradual change? The answer to this question cannot be given once and for all. This is not only due to the complexity of human perception and cognition, but also because the answer depends on size and complexity of the considered dataset. Nonetheless, it is possible to give some hints on the use of actions and processes.

Since actions result in instant changes of the visual representation, they do attract the attention of users. If an action achieves a "pop-out"[6] of relevant parts of a visual representation, users conceive immediately not only that an interesting event occurred, but also where in the data it happened. On the other hand, it is the goal of every visualization to establish a mental map of the data in the users brain. Current visualization techniques achieve this goal only for small datasets where the relevant information fits to the screen. For large and complex datasets, different and/or multiple views are required to establish a mental map of the data. If, in such cases, the visualization changes instantly, the mental map is hard to maintain, and users are likely to lose the overview of the data. When looking at gradually changing representations, users can apply the visual changes more easily to their mental maps. Processes can also be used to support the short term memory of users during dynamic visualizations. For doing so, processes can be implemented such that they fade out event representations gradually over time. By doing so, the visual stimulus is preserved for the time of the fading, and thus the users are provided with more time to react to or memorize the relevant information. Furthermore, gradual changes are helpful in supporting the navigation in the data, e.g., by using animation to switch to a viewpoint that shows an event and its immediate data context. Therefore, in cases where it is essential to have a mental map (e.g., when visualizing abstract data like graphs), it would be better to use processes, which apply changes gradually, rather than instantly. However, when using processes, it must be kept in mind that they consume time. For this reason, it must be ensured that processes realize an emphasis on event instances before a successive event detection is conducted. This is essential especially if events must be detected frequently (e.g., for highly dynamic data). If processes did not accomplish the event representation in between two successive event detections, users would not be able to perceive the intended changes in the visual representation. This would contradict to the established requirements.

Actions and processes both aim at emphasizing special portions of the data or interesting situations during the visualizations. To achieve this, they apply changes to the visual representation. Such visual changes can be either:

- Local or
- Global.

[6] The "pop-out" effect is a term known in cognitive psychology meaning that certain objects of some visual representation are perceived preattentively (they pop-out), whereas others are not [Ware, 2000].

Local changes emphasize only portions of the screen to achieve the requirements of the event representation, i.e., they have a local scope. An example for a local adaptation is the change of color and width of a line segment that represents a maximum value (see Figure 4.12 on page 81). When speaking of global changes, it is meant that the visual representation is affected as a whole: Actions or processes that apply global changes cause the visual representation to be perceived entirely different. For instance, an action that switches between completely different color scales or a process that moves the focal point of a globally effective fisheye distortion on occurrence of certain events are considered as changes with global scopes.

Again, the distinction between local and global changes makes it necessary to discuss the question when to use local or global changes. Local changes, on the one hand, are limited in their spatial extension on the screen, and hence enable the representation of multiple event instances in a single view. Furthermore, the locality of the changes leaves space for conveying the type of event instances. On the other hand, local changes, especially many of them, might be hard to grasp for users, because local changes compete with each other for the users' attention. If the application scenario allows attributing different events with different importance, it is possible to account events that have a higher importance with higher priority in the visual representation. Apparently, this requires the implicit event representation to be scalable. In the case of a binary importance classification (i.e., prioritized events vs. common events), the scalability is not difficult to achieve. For discrete or even continuous importance functions, it is a challenging task to accomplish the scalability of the event representation. Nonetheless, a prioritization of events can help to control how local changes of the visualization should attract the attention of users. Global changes, on the other hand, do not compete for the users' attention. Since global changes affect the whole display, rather than only portions of the screen, they are useful for conveying events that abstract from concrete data items (e.g., composite events). However, the global scope of the changes also implies that only a few events can be incorporated into the visualization simultaneously. To achieve changes that are globally distinguishable, actions or processes must address different channels of perception (e.g., if a global action addresses perception of color by changing a color scale, then another global action could address the spatial perception by changing the layout of the visualized data).

From the previous discussion it becomes clear that finding expressive implicit event representations is a challenging task. Particularly the diversity of application scenarios, which require different ways of considering events in the visualization, makes this task demanding. As indicated, also perceptual issues play an important role for implicit event representations. For more information and references on the cognitive aspects touched in the previous discussion, the interested reader is referred to [Treisman and Gelade, 1980], [Healey et al., 1996], [Zhai et al., 1997], [Bartram et al., 2001], [McCrickard et al., 2001], [Somervell et al., 2002], and [Tversky et al., 2002].

How to Compile Actions and Processes

Now that actions and processes have been introduced, it will be described how actions and processes can be specified. A first possibility would be to develop a formal language for specifying actions and processes with respect to arbitrary visualization techniques (similar to the event specification with respect to the relational data model). This would enable users to define freely how to integrate events into a visualization. However, what renders this idea hard to realize is the lack of a formal model of interactive visualization techniques. Such a model is required prior to specifying parameter changes as actions or processes. Due to the complexity of interactive visualization techniques, the development of formal foundations is still a current challenge of research in the field of Visual Languages (e.g., [Bottoni et al., 1995], [Bottoni and Costagliola, 2002]) and the emerging field of Visual Analytics (see [Thomas and Cook, 2005]). As of the writing of this thesis, there are no mature approaches available to describe interactive visualizations formally. This means that there is no basis for a formal language for specifying actions and processes. Moreover, it is questionable if such formal language can be operated easily, or if it would not be the case that users would even need programming skills to be able to define actions and processes on their own.

Therefore, it is suggested that visualization experts should be in charge of designing and implementing implicit event representations for different visualization techniques. As a first step, it is necessary to analyze which parameters are offered by a particular visualization technique, and how these parameters can be adjusted to achieve the stated requirements of the event representation. In a second step, the developed actions and processes must be assigned to event types. In this regard, it is important to match the event representation with the needs of the addressed application scenario and user group. In particular, it should be evaluated whether actions and processes, more precisely, the resulting local and global changes, generate the intended effects. To achieve the best visual results, users and visualization designers have to work closely together on this matter.

Finally, the outcome of the previously described procedure are actions and processes that are tailored to the considered visualization technique and application scenario; the particular actions and processes have become an integral part of that visualization technique.

Actions and Processes for Implicit Event Representation

The goal of this section is to illustrate possibilities to implement implicit event representations via actions and processes. The suggested parameter adaptations are not tailored to particular visualization techniques, but are intended to be general-purpose to allow their easy adaptation to any visualization technique that provides adequate visualization parameters.

The following paragraphs will follow the steps of the visualization pipeline (see Section 2.1). These steps include the *data analysis* step for preparing raw data for the visualization, the *filtering* step for selecting data portions to be visualized, the *mapping* step for mapping data to geometric primitives and their attributes, and the *rendering*

step for creating the final visual representation [dos Santos and Brodlie, 2004]. Even though the visualization pipeline is a useful tool for describing visualization in general, it provides only a coarse categorization of the visualization transformation. Therefore, the association of particular steps of the visualization pipeline with actions and processes as described in the following is arguable and not meant to be definite.

Adapted data analysis Since raw data does not always meet the requirements of a visualization technique, it is the task of the data analysis to perform the necessary steps for mapping raw data to data that is prepared for the visualization. The data analysis step performs data-centric operations like data smoothing, extrapolation of missing values, or clustering.

The data analysis step comprises mostly statistical and analytical methods. Such methods usually provide various parameters that need to be set with care to create the intended results. Although one could argue that it is the task of the users to set these parameters adequately, for an effective visualization it is indispensable to support users in this task. By developing actions that adapt the parameters (e.g., of a clustering algorithm) automatically according to the occurrence of events that indicate certain properties of the data, such a support can be realized. Since statistical or analytical methods do not generate immediate results, but need a certain amount of time for the calculation, it is suggested to implement parameter changes as instantaneous actions. Actions can also be used to simply trigger the execution of specific methods in the data analysis step. If, for instance, an event detects a time-dependent attribute whose values fluctuate very much (i.e., an attribute with a high frequency), an action can be used to trigger a method to smooth the respective attribute. Time varying processes are impractical, because they require repetitive calculations to generate intermediate results for changing the visualization gradually.

In this sense, automatic parameter changes can be very useful in the data analysis step. However, most of today's visualization techniques do not consider the data analysis step explicitly, but implicitly perform some statistical calculations throughout the visualization pipeline; only a few visualization techniques provide adequate parameters that can be adjusted via actions. This corroborates the need for a tighter coupling of analytic and visual methods as pursued in visual data mining [Keim, 2002] and the emerging field of Visual Analytics [Wong and Thomas, 2004].

Before advancing to implicit event representations at other steps of the visualization pipeline, it must be stated that by applying parameter changes exclusively with respect to the data analysis step, it is practically impossible to achieve the requirements of the event representation. To realize expressive event representations, considering events in the data analysis step is always supplemental to considering the same events in at least one of the further steps of the visualization pipeline.

Adapted filtering The filtering step serves as an excellent entry point for semantically focusing on interesting parts of the data. Two filtering operations are essential in the context of relational data – *selection* and *projection*. A selection imposes a restriction on the tuples to be visualized, whereas a projection restrains the attributes to consider.

For the purpose of implicit event representation, selections and projections should be implemented as actions. Processes are not recommended, since computing selections and projections might be costly in terms of computation time, depending on the volume of the considered data.

Selections and projections are particularly useful to filter out tuples and attributes for which no events have been detected, i.e., only the most relevant portions of the data are passed on to the next steps of the visualization pipeline. To also convey the context in which events occurred, it is sometimes not desirable to apply a strict focusing, but to consider an adequately sized neighborhood of the data elements of interest. For a selection, the decision of which tuples belong to a neighborhood can rely on the calculation of similarities[7] between tuples that are linked to event instances and all other tuples in the data. To determine a neighborhood for a projection, it makes sense to calculate similarities[8] between attributes that are linked to events and all other attributes. Tuples and attributes with higher similarity and correlation factors, respectively, are considered to belong to the context that should be visualized as well.

Since less-relevant tuples and attributes (with respect to the user-defined event types) are neglected and only data elements being linked or being similar to event instances are passed on to the remaining steps of the visualization pipeline, applying automatic selections and projections is particularly useful to reduce the volume of data to be considered when visualizing large datasets.

Adapted mapping The mapping step transforms filtered data to geometry (e.g., points, lines) and associated properties (e.g., color, width), and secondly, lays out the geometry spatially to create a visual representation. Since the mapping step is the most crucial one to achieve the visualization requirements (i.e., expressiveness, effectiveness, and appropriateness), it is also the most important one for achieving the stated event representation requirements. In general, most visualization techniques provide a variety of parameters to control the mapping step interactively. However, an automatic adjustment of the parameters is pursued only rarely.

Possible automatic parameter changes are introduced next. To ease the understanding of the manifold ways of automatically adjusting the mapping step, the following categorization is suggested:

- Adaptation of the geometry,
- Adaptation of the geometry's properties, and
- Adaptation of the spatial layout of the geometry.

Adapting the geometry is particularly helpful in achieving the requirement for highlighting of event instances. This becomes clear when considering a visualization that usually encodes all data items to the same type of geometry. If now the geometry is changed for data items that are associated with events, relevant information is immediately perceived among the remaining homogenous geometry; the changed geometry

[7] Similarity among tuples can be computed based on distances in the value ranges of the data.
[8] Similarities between attributes can be computed by applying, for instance, [Ankerst et al., 1998].

attracts the attention of users. The geometry type is also useful to communicate different event types. In other words, apart from the geometry used to represent common data items (e.g., a circle), each considered event type can be associated with a different type of geometry (e.g., triangle, rectangle, line). However, this is possible only for a limited number of event types. Finally, focusing on relevant data can also be facilitated by adapting the geometry type: If an event has been detected, it makes sense to switch from a simple geometry to a more detailed one that allows a closer analysis (see [Matković et al., 2002a]). Although the previous lines indicate instant switches of the geometry (actions), it is also possible to consider smooth transitions between geometry types (processes). In the latter case, algorithms are needed to smoothly transform (morphing) between different types of geometry (see e.g., [Ebert et al., 1997]). Such gradual transitions would support the perception and memorization of the represented events.

As a second possibility, the properties of the geometry can be adjusted in the mapping step. Basically, representational properties like fill or stroke color, fill or stroke opacity, stroke width, or stroke dash pattern are subject to automatic adaptations. Color is an important property to represent events implicitly. By adjusting the colors used in a visualization, interesting data elements or relevant value ranges can be highlighted easily (see [Rogowitz and Treinish, 1998] or [Tominski et al., 2005c]). Alternatively, it is possible to emphasize interesting data elements by fading out less-relevant ones. In other words, only relevant information is drawn in full saturation, whereas less-relevant information is drawn in reduced saturation or is even hidden from the user (information hiding). This can be accomplished by adjusting the translucency of the geometry used. Although it is suggested to apply immediate changes for purposes of highlighting (e.g., instantly switch to a highlighting color), information fading should be applied in a continuous fashion, so that users are aware of the fading. Another advantage of using processes for fading is that possibly occluded information is uncovered gradually such that pop-up artifacts, which would result in an undesirably strong attraction of the user's attention, are avoided.

Whereas the previously described mapping adaptations address geometric primitives and their properties, the third option considers the layout of the geometric primitives on the display. The layout of the geometry has a major impact on the expressiveness of the event representation. To adapt a given layout, it makes sense to take the positions and the size or extent of the geometric primitives into account. Even though position and size are properties of the geometry that can also be altered independently, when speaking of layout adjustment, it is meant that usually multiple positions and sizes must be changed as a whole to achieve effective event representations. Although layout adjustments are mostly of global nature, i.e., affect the whole visualization, it is also possible to locally adapt a given geometry layout. As an example for layout adjustments one can imagine a focus+context technique[9] that brings interesting data portions to an exposed location of the display (e.g., the display center) and provides them with more display space to make them visually more present. Less-important context information can be encoded in the periphery of the display using less space. The position of the

[9]Such techniques are illustrated in Chapter 5 and Chapter 6.

focal point, which determines which data portions are focused, can be adjusted as a response to an event occurrence. Due to the mostly global character of layout changes, it is suggested to use processes for their realization. By doing so, the mental map of the presented information can be maintained more easily, compared to an instant switch in the whole layout.

A variety of event representations can be accomplished using the previously explained options for adapting the mapping step of the visualization pipeline. Additionally, a further category of event representations, which is linked to the mapping step, shall be illustrated. Specifically, it is suggested to integrate additional visual cues to emphasize interesting data portions. Such visual cues could be either

- Markers or
- Labels.

Markers are graphical primitives that can be used to emphasize some geometry. In other words, markers do not encode data elements, but point out geometry that encodes relevant data elements. Markers could be, for instance, arrows pointing out interesting aspects of a visual representation or circles surrounding event-related data elements. To control markers, parameters like marker type, position, orientation, or color can be adjusted by appropriate actions. Similar to markers, labels also support the emphasis on relevant data elements. Labels, however, use alphanumeric information (e.g., annotation or event descriptions) to achieve the highlighting. Using actions to place labels on relevant data elements will draw the users' attention toward these parts of the data. In conclusion, markers and labels are particularly useful in scenarios where the visual presentation[10] of relevant information is the major objective.

Adapted rendering At the end of the visualization pipeline, the rendering step is the last stage that can be adapted to create expressive event representations. In the rendering step, standard methods of Computer Graphics are used to transform the geometry created in the mapping step into images that can be displayed on a computer. The parameters of these methods provide several options to adjust the rendering by actions or processes.

Specifically, it makes sense to adapt the viewport of the virtual camera, i.e., parameters like camera position, orientation, or field of view. As mentioned in Section 2.1, overview+detail techniques are commonly used to support the visualization of large datasets. This approach implies that users have to switch frequently between overview and detailed views. Automatic viewport adaptations are useful to reduce the number of navigation steps that would have to be performed manually. This means that on occurrence of an event, the viewport is automatically relocated such that the visualization shows the portion of the data in which the event occurred. To implement viewport changes it makes sense to use gradual processes to support users in maintaining their mental map of the data. How viewports can be switched in a smooth and efficient way is described, for instance, in [van Wijk and Nuij, 2004].

[10] One of the application scenarios of visualization (introduced in Section 2.1).

A limitation of viewport adaptations is that it is only possible to focus the users' attention on a single event. If multiple events must be considered, other event representations are required. Decreasing the visual presence of less-relevant data portions (i.e., parts for which no events have been detected) is a possible solution to this problem. For doing so, the geometry representing less-relevant data can be rendered with a lower degree of opacity. It is also possible to adjust the depth of field (DOF), to blur the geometry that encodes less-relevant data [Kosara, 2001]. By both methods, unimportant information is melted with the background of the visualization and only the geometry representing relevant information (i.e., data portions associated with events) is rendered opaque and sharp. As evaluated in [Kosara, 2001], this enables users to perceive relevant information preattentively. Therefore, it can be stated that adjusting opacity and depth of field are very useful methods to achieve the first and second requirement of the event representation.

Because many visualization techniques cannot guarantee that one pixel of the display represents only one data item, occlusion of information is commonly an issue. It is likely to happen that geometry encoding important data items must share some pixels of the display with other geometry. To ensure the visibility of relevant information, it is necessary to adjust the drawing order in the rendering step: Geometry representing less-relevant information must be rendered first, whereas geometry representing data portions associated with events must be rendered at last. By doing so, occlusion of geometry representing events can be avoided. When considering opacity changes to dilute the visual presence of less-relevant information in three dimensional presentation spaces, also the distance between geometry and virtual camera must be taken into account to determine the correct drawing order. In any case, the way of how geometry is rendered is an important aspect to consider when designing event representations.

The rendering step can also serve as an entry point to increase the performance of visualization techniques. Especially for large datasets it is challenging to achieve interactive rendering times. To provide users with immediate visual feedback to their interactions, it makes sense to render only parts of the data during interaction and to process the dataset as a whole in a separate background thread. A difficulty of this procedure is to decide which parts of the data should be rendered and which should be omitted during the interaction. A random sampling of the data would be a simple, but not always expressive possibility. Another solution is to exploit the knowledge that events occurred in the dataset. The rendering parameters should be adjusted in such a way that only geometry representing event-related data is rendered during interaction. If this restriction is too weak with respect to the available computational power (i.e., still too much geometry is considered), only geometry representing events with higher importance should be rendered during interaction. If, on the other hand, considering only events is too conservative (i.e., the computational power is not fully exploited), certain neighborhoods of the event-related data portions can be considered. With a rendering step realizing the previous suggestions, it is possible to ensure that the most relevant information can be represented at interactive response rates. If the rendering step is very expensive in terms of computational efforts, it also makes sense to integrate more sophisticated methods like, for instance, importance-driven rendering

(see [Schultz and Schumann, 2001], [Schultz, 2002]). In three dimensional visualization environments that use multi resolution approaches it would also be possible to adjust the level of detail according to occurred events.

Using lenses for implicit event representation In the previous discussion, diverse implicit event representations have been described with respect to the different steps of the visualization pipeline. In this section, an additional interesting means for implicit event representation is described – computer-generated lenses, which can operate on virtually any stage of the visualization pipeline. In reality, real lenses are usually applied to look at interesting things in detail. Accordingly, lenses can also be applied in computer-generated visual representations to analyze interesting portions of the data in detail. Even though the usefulness of such lenses was already described in 1994 (see [Stone et al., 1994]), only few of today's visualization approaches consider the advantages of lens techniques. Since lenses can also be applied to guide the attention of users, they are particularly useful in event-based visualization.

An interactive lens is usually described by a focal point and an arbitrary shaped two or three dimensional area. This area defines the scope of the lens and is either superimposed on or embedded in a visual representation of some data. Since a different view of the data is provided within a lens' scope, lenses can be understood as visualizations within visualizations (or second order visualizations [Björk et al., 1999]). Lenses can operate at any stage of the visualization pipeline, or even implement all stages. This allows a distinction of lenses according to the steps of the visualization pipeline into semantic lenses (operating on the data-centric steps), mapping lenses, and graphical lenses. Lenses that combine multiple basic lenses at one or more stages are called hybrid or composite lenses.

An advantage of lenses (besides the ability to provide a different view of the data within the original visualization) is that they can be easily controlled by the two parameters – focal point and lens scope. This makes it easy to develop actions or processes to focus on relevant data portions of a dataset. For semantic lenses, the focal point can be set on a data item associated with an event, whereas for mapping lenses or graphical lenses, the focal point is set with respect to the geometry representing interesting data portions. Accordingly, the lens scope can be adjusted to comprise certain neighborhoods of the relevant information. Although commonly only one lens is used for a detailed analysis, depending on the concrete design of a lens technique, it is also possible to use multiple lenses simultaneously. This allows the emphasis on multiple data items that are associated with events. However, it must be mentioned that in some cases the scopes of the lenses must not overlap each other. To solve this issue, the scopes of the lenses can be scaled down until they obey this restriction, or only one lens with an enlarged scope can be used to highlight multiple interesting data items as a whole.

In recent years, several researchers have proposed a variety of helpful interactive lens techniques for different visualization tasks and different application scenarios (e.g., [Stone et al., 1994], [Rao and Card, 1994], [Rase, 1997], [Keahey, 1998], [Shaw et al., 1999b], [Griethe, 2004], [Ellis et al., 2005], or [Tominski et al., 2006a]). In contrast to

other focus+context approaches (e.g., [Rauschenbach et al., 2000]), these lenses work on spatially confined regions instead of being applied to the whole visual representation. If any of these lenses are available in a given visualization system, they should definitely be exploited to emphasize on event-related data portions. In Chapter 6, the usefulness of lenses for visualizing large graphs will be demonstrated.

Summary on Implicit Event Representation

The preceding descriptions indicate that throughout the visualization pipeline a variety of options do exist that can be used to achieve expressive event representations. This variety underlines the requirement that visualization designers should be in charge of specifying adequate actions and processes for the event representation; common users would be overwhelmed by this task. Moreover, it is not fully determined in which step of the visualization pipeline certain options can be applied. For example, it is possible to use transparency to reduce the visual presence of less-relevant data portions either in the mapping step or in the rendering step. In this case, the visualization designers have to make the final decision depending on the concrete visualization technique.

Another aspect that becomes clear when looking at the described possibilities to represent events concerns the use of actions and processes. For data-centric steps of the visualization pipeline (data analysis, filtering), it is suggested to use actions. The reason for this is that data-centric methods usually require higher computational effort, which renders gradual changes hard to implement. On the other hand, processes are particularly useful in the graphics-related steps of the visualization pipeline (mapping, rendering). In these steps, gradual changes can draw the attention of the users to relevant data portions and at the same time they support users in maintaining their mental map of the data.

4.6.3. Explicit Event Representation

In Section 4.6.2 it has been explained how events can be visualized implicitly by merging their representation into known visualization techniques. In contrast to that, this section is dedicated to the explicit representation of event instances. It is described what the aims and requirements of explicit event representation are, and suggestions are made on what techniques are suitable for explicit event representation.

When speaking of explicit event representation it is meant that not a dataset D, but only the event instances $E = \texttt{detect}(ET_{relevant}, D)$ detected in the data are visualized ($ET_{relevant}$ denotes the set of relevant event types), i.e., the input to the visualization transformation are only the detected events. According to [Reinders et al., 2001], such a focusing on relevant events raises the level of abstraction of the data analysis. Although visualizing only events is contradictory to Tufte's claim *"Above all show the data!"* [Tufte, 2001], providing a more abstract event representation (certainly in addition to a fine-grained data visualization) is useful for achieving more complex visualization tasks like finding correlations between different interesting aspects of the data (described as event types). Therefore, it can be stated that the major aim of representing events explicitly is to facilitate higher order visual analysis. A second

matter that is definitely important to mention concerns the fact that commonly the number of detected event instances is much smaller than the number of data items $|E| << |D|$. This fact can be exploited to alleviate the visualization of very large datasets. By relying on events only, rather than on the whole dataset, insight into even large datasets can be gained. Therefore, the second major rationale behind explicit event representation is coping with large volumes of data.

To achieve the identified goals, it is suggested to transform the event instances detected in some dataset into a new dataset that, in turn, is subject to a visual analysis. Formally, this transformation can be modeled as a mapping

$$\texttt{eventToData} : \mathfrak{P}(\mathcal{E}) \rightarrow \mathcal{D}.$$

Even though different application scenarios might require different implementations of this mapping, all should have in common that applying the mapping translates a set of event instances E to a new relational dataset $D' = \texttt{eventToData}(E)$ that can be visualized directly. In addition to that, since the event type is a crucial characteristic, any implementation of `eventToData` should consider the event type as an attribute in D'. Further event parameters like event importance should also be mapped to attributes in D'.

The advantage of this procedure is that a broad range of available visualization techniques can be used to represent the event instances E (now mapped to the new dataset D') explicitly. Since the event type is a categorical (i.e., nominal) attribute, it is a major requirement that the visualization techniques are capable of efficiently representing categorical data. Since it might be necessary to visualize more than one event parameter, potential visualization techniques should also be able to handle multivariate data.

Depending on the implementation of the `eventToData` transformation, different ways of representing events explicitly are possible:

1. Representation of abstract event instances,
2. Representation of event instances in time, and
3. Representation of event instances in time and space.

With respect to this distinction, different possibilities of explicit event representation are discussed in the following paragraphs.

Representing abstract event instances In this first case, event instances are interpreted as abstract entities that have associated certain parameters. Accordingly, D' can be considered a relational dataset of the form

$$D' \subseteq ET \times EP_1 \times \cdots \times EP_k$$

where ET denotes the event type and $EP_i \mid 1 \leq i \leq k$ denote different event parameters (e.g., event importance). However, relying on the event type and possibly additional event parameters is barely sufficient for any visual event representation. Therefore,

it makes sense to enhance this abstract interpretation with statistics derived from D' (e.g., absolute and relative event frequency in D'). It is then possible to apply classic visualization techniques in order to visually relate event types (and possibly additional parameters) to derived statistical characteristics. Simple diagram techniques [Harris, 1999] can be used, for instance, to visualize frequencies of event instances (histograms for absolute frequency, pie charts for relative frequency). As already indicated, it is crucial to represent the event type, which is a categorical attribute in D'. Therefore, it is mandatory that the visualization technique either supports categorical data natively or that methods are applied that assign a numeric value to each category, i.e., to each event type (e.g., [Rosario et al., 2004]). Even though these first explicit event representations are quite abstract and simple, they are nonetheless an initial step toward higher level insight into the underlying data. The drawback of representing event instances as abstract entities is that the frame of reference of data and events is not considered, which results in an increased difficulty of creating expressive event representations. Hence, the understanding of the generated images is not always intuitive.

Representing event instances in time On the other hand, events are virtually always connected to time. Particularly, composite events as introduced in this thesis are anchored in the (external) visualization time. Depending on the characteristics of the visualized dataset, also tuple events and sequence events could be connected to (internal) time. Therefore, it makes sense to take time as a reference dimension into account to be exploited for explicit event representation. This means that a set of event instances is mapped via `eventToData` to a relational multivariate temporal dataset that consists of a time attribute $\mathfrak{T}$ and attributes describing the events

$$D' \subseteq \mathfrak{T} \times ET \times EP_1 \times \cdots \times EP_k.$$

This transformation enables using visualization techniques for time-related data to represent event instances. Since visualization of time-related data is a challenge in its own right [Aigner, 2006], only a compact overview of possible event representations will be given in the following.

A first and commonly accepted differentiation that must be made when visualizing data (or events) in time concerns the basic temporal elements used for modeling the temporal domain. Basically, a temporal domain is made up of either time points or time intervals[11]. This distinction implies that dedicated visualization techniques have to be used to represent differently interpretable events. In other words, if the interpretation links events to time points (e.g., for tuple events or composite events), different techniques must be used than those used when events are related to time intervals (e.g., for sequence events). Although further categorizations of time are suggested in literature (e.g., in [Frank, 1998] distinctions are made between linear, cyclic, or branching as well as between discrete or continuous time domains), these categorizations are not

[11] Since it can be assumed that the reader has a natural understanding of the terms time point and time interval, a formal description will not be presented here; interested readers are referred to [Hajnicz, 1996] or [Aigner, 2006].

considered with respect to event representation. The reason for this is that these categorizations are barely considered by today's visualization techniques [Aigner, 2006]. More efforts have to be spent on developing appropriate visualization techniques for these types of time. However, this is beyond the scope of this thesis. Nonetheless, being aware of different types of time is important for creating expressive visualizations of events in time. Particularly the differentiation between time points and time intervals must be taken into account.

Until today, most techniques for visualizing time-related data support the visual representation with respect to time points. From simple yet expressive diagram techniques [Harris, 1999] to more advanced techniques like [Carlis and Konstan, 1998], [Hochheiser, 2003], or [Tominski et al., 2004] a broad range of tools is available that can be used for representing events with respect to time points. If such techniques are applied, usually the most important event characteristic, namely the event type, should be chosen for visualization. As described in [Reinders et al., 2001], representing alongside time, the course of occurred events facilitates high-level insight into the analyzed data.

If events are bound to time intervals, other techniques should be applied. In recent years, several researchers have investigated the visual representation of time intervals (e.g., [Cousins and Kahn, 1991], [Kosara and Miksch, 2001], or [Chittaro and Combi, 2003]). Even though their research was mainly driven by the need for tools to reason about time, the developed techniques can also serve as a basis for representing events bound to time intervals.

(a) (b)

Figure 4.14.: Examples of suitable techniques for representing events in time.
(a) ArcDiagrams [Wattenberg, 2002]; (b) Spiral display (e.g., [Weber et al., 2001]).

A natural goal when representing event instances with respect to time is to reveal higher level relations between them, and by this, to find correlations in the underlying dataset. How this goal can be accomplished is illustrated by the example of two commonly accepted visualization techniques. The first of them are Arc Diagrams [Wattenberg, 2002] (see Figure 4.14 (a)). This technique was originally invented to visualize patterns in strings. To compute an Arc Diagram, essential patterns of substrings are automatically derived from a character sequence, and occurrences of the same pattern

are visually connected using *...thick semi-circular arcs* [Wattenberg, 2002]. Similarly, it would be possible to detect patterns in a course of detected events. For doing so, an event course is interpreted as a sequence of characters in which each event type is represented by a unique character. The Arc Diagram technique can then be applied as usual, resulting in a representation of high-level relations between events (i.e., the interesting aspects of the data). In contrast to Arc Diagrams, which can be used to perform an automatic detection of possible relations between events, the second technique that can be applied to analyze event courses follows an interactive approach. Spiral displays (e.g., [Carlis and Konstan, 1998], [Weber et al., 2001], [Dragicevic and Huot, 2002], or [Tominski et al., 2005b]) are interactive visualizations that represent the time domain by mapping it onto a spirally shaped time axis (see Figure 4.14 (b)). Depending on the particular implementation, spiral displays can represent data (in this case the event type) with respect to time points or time intervals. The major advantage of spiral displays is their capability of representing periodic patterns. From the event representation point of view, this opens up the chance to identify recurring patterns of events. To enable the detection of periodic patterns of different length, spiral displays commonly allow users to interactively adjust the number of temporal primitives (either time points or intervals) encoded per cycle of the spiral time axis. Additionally, the number of cycles used to create the spiral can be interactively altered. This enables browsing and exploring the event course for possible event patterns. Both of the suggested concepts for representing events explicitly are easy to implement, and thus can be easily incorporated into an event-based visualization system.

Considering events in a temporal domain increases the expressiveness of explicit event representations. Automatic or highly interactive approaches can be useful to facilitate the analysis of event occurrences. Nonetheless, the time attribute is the only reference dimension according to which events can be analyzed. In the next paragraph it is described how additional spatial reference dimensions can be addressed in the event representation.

Representing event instances in time and space Even though considering time as the only reference dimension for explicit event representation might be sufficient in certain application scenarios, it is more often the case that also spatial reference dimensions have to be taken into account. In particular, spatio-temporal data and events derived from such data are highly relevant in many application fields. Therefore, it makes sense to visualize spatio-temporal events with respect to a spatio-temporal frame of reference. For this case, the `eventToData` mapping is supposed to create a dataset of the form

$$D' \subseteq \mathfrak{T} \times \mathbb{R}^n \times ET \times EP_1 \times \cdots \times EP_k$$

where $\mathfrak{T}$ denotes the temporal frame of references and $\mathbb{R}^n$ denotes the spatial frame of reference. Since the spatial frame of reference is usually related to the physical world, it comprises $n = 2$ or $n = 3$ dimensions.

In addition to being able to handle categorical and multivariate data, visualization techniques that should be used to represent spatio-temporal events must also be able to cope with multiple independent dimensions. Similar to what was said for visualizing

events in time, the representation of multivariate data in multidimensional spaces is a challenge addressed by many researches [Wong and Bergeron, 1997]. The difficulty of creating expressive visualizations of such data stems from the fact that multiple attributes must be mapped to a two-dimensional output device. Although techniques that create visualizations of datasets with multiple attributes, irrespective of whether they are independent dimensions or dependent variables, do exist (e.g., Parallel Coordinates [Inselberg, 1998] or Scatter Plot Matrices [Cleveland, 1993]), most of these techniques represent the frame of reference only in an abstract way. To facilitate the understanding of event occurrences in time and space, the more desirable solution is to represent the frame of reference in close correspondence to the real word, rather than abstractly.

This solution implies two requirements for the explicit representation of spatio-temporal events:

1. The frame of reference must be visualized.
2. The occurred events have to be represented within the frame of reference.

To represent an abstract spatial frame of reference ($\mathbb{R}^2$ or $\mathbb{R}^3$), a two- or three-dimensional presentation space (2D or 3D) can be used as usual. In this case, the presentation space reflects the reference space in direct correspondence. If the considered data have been collected in a geographically-related environment, maps are an excellent choice to represent the spatial frame of reference. This claim is corroborated by the fact that maps have been used for ages to illustrate information with respect to geographic realties. Maps have the advantage that they allow the representation with respect to natural or manmade spatial phenomena like continents or states. This facilitates the understanding of represented data and events.

To represent the temporal dimension of the reference space, it is possible to use animation techniques, i.e., to map the temporal dimension to the physical time. Even though this results in a one-to-one correspondence between the temporal reference dimension and its representation, animation approaches bear some inherent drawbacks. These drawbacks have already been discussed in Section 3.1 (also see [Tversky et al., 2002]). Alternatively, the temporal dimension can be mapped to a dimension of the presentation space. In the case of a 3D presentation space, two dimensions represent the spatial frame of reference and one dimension encodes time. If three spatial as well as one temporal dimension must be represented, there is, except of using animation, no other way than superimposing dimensions of the reference space in the visual representation.

If a representation of the frame of reference has been found, it is possible to visualize the occurred event instances. This can be achieved by using glyph representations of event instances. Glyphs are small graphical primitives that encode multiple data attributes (see e.g., [Ribarsky et al., 1994], [Shaw et al., 1999a], or [Kreuseler and Schumann, 1999]). Analogous, it is easily possible to map the event type and further event parameters directly to glyphs. In addition to that, glyphs can be arbitrarily arranged on the display, by which an additional degree of freedom is available for encoding information. In particular, glyphs are positioned in the presentation space at points that correspond to points of the frame of reference where events occurred,

allowing users to identify clusters of spatio-temporal events. The course of events in time and space can also be made explicit by utilizing the space-time path approach. Space-time paths have a long history in cartography, where they have originally been used to represent human activity [Carlstein et al., 1978]. Basically, space-time paths depict trajectories of objects through time and space. They are also applicable to illustrate trajectories of events. For doing so, glyphs representing successive events of the same type are connected visually via a line segment. Doing this for all events and all event types results in a set of space-time paths that represent event trajectories. This kind of representation facilitates the understanding of relations between events and time and space as well as between events of different types.

The space-time paths as illustrated here have been incorporated into the event-based visualization framework eVis, which is introduced in the next chapter (see Figure 5.19 on page 139).

Summary on Explicit Event Representation

In this section, a last aspect of event-based visualization has been described – the explicit event representation. It has been pointed out that no new visualization techniques are required for this purpose. Only the mapping $D' = \texttt{eventToData}(E)$ must be implemented to derive a new dataset D' from a set of detected event instances E. This new dataset D' can then be visualized using most of today's visualization techniques. Since the event type, which is mandatory to visualize, is a categorical value, the visualization technique used must be able to handle categorical data. Several techniques that meet this requirement have been suggested. Furthermore, it has been explained that considering events in time can facilitate a deeper insight into the data. Visualizing event instances in a spatio-temporal frame of reference can help revealing interesting relations between events and the space in which they occurred. Finally, since the new dataset D' is usually smaller than the dataset D from which the events have been detected ($|D'| << |D|$), it is also possible to gain insight into large datasets.

4.7. Chapter Summary

This chapter was dedicated to the introduction of a general model of event-based visualization. In the first section, unsolved problems of current visualization approaches have been described and the goals to be achieved in this thesis have been identified. These goals have been addressed with the development of a novel generalized model of event-based visualization. An informal illustration of the central parts of this model – the event specification, the event detection, and the event representations has been substantiated with a formal definition of event-based visualization. With respect to the event specification, the notion of abstract event types has been introduced and different abstract event types that address relational datasets have been described formally. It has also been explained how users can be supported in defining events they are interested in. In another section, formal means for detecting event instances have been introduced, and methods for improving the detection efficiency have been mentioned.

In the last section of this chapter the most important issue of event-based visualization – the event representation has been discussed. On the one hand, implicit event representation has been suggested as a means for incorporating visual representations of detected event instances into given visualization techniques. The basic idea of implicit event representation is to automatically adjust visualization parameters according to occurred events. Several possibilities to automatically adjust visualization parameters at different stages of the visualization pipeline have been proposed. On the other hand, explicit event representation has been suggested as a means to facilitate higher level visualization tasks. This is achieved by leaving aside the original data, and representing only the event instances that have been detected in the data.

All proposed concepts are subject to further investigation in the future. With respect to the event specification, the expressiveness of the suggested abstract event types must be examined. Particularly, the suggested definition of the abstract composite event type can be improved using regular expressions. The suggestions for future work on the event detection can be summarized as a general need for efficient detection algorithms. Incremental event detection has been identified as a promising approach to achieve performance improvements.

Whereas in this chapter the event representation has been described on a conceptual level, the next chapter will illustrate concrete examples of implicit and explicit event representations implemented in the event-based visualization framework eVis. A discussion of future improvements of the event representation will also be given in the next chapter.

5. eVis – An Event-Based Visualization Framework

In the previous chapter, the concept of event-based visualization has been introduced. To substantiate the developed concept, this chapter explains how event-based visualization can be implemented on a computer system. Based on a requirements analysis, the architecture of the framework *eVis* is introduced. eVis has been developed in accordance with the generalized model of event-based visualization, and as such, involves the three main aspects event specification, event detection, and event representation. Besides giving brief descriptions of the basic components of eVis, this chapter will detail on a visual event specification environment that aims at supporting users in specifying event types. To demonstrate the potential of the concepts introduced in this thesis, several visualization techniques are described that implement event-based visualization as proposed in Definition 4.1 (p.49). In particular, it is explained how the well-known TableLens technique [Rao and Card, 1994], novel axes-based techniques created with the toolkit *VisAxes* [Tominski et al., 2004], and map-based representations generated with the *TimeMap* toolkit [Tominski et al., 2005b] can benefit from integrating them into the framework eVis.

5.1. Requirements Analysis

Designing a visualization framework has different requirements and challenges than developing just a single visualization technique. A visualization technique is usually tailored to a particular dataset and visualization problem, whereas a visualization framework must be suitably flexible to address various visualization problems and data from diverse scenarios. To successfully develop a visualization framework, a thorough requirements analysis is necessary. Such an analysis has been conducted, of which the key requirements are illustrated in this section. Besides general requirements for a visualization framework, additional event-related requirements will be explained.

Data As has already been stated in the basics chapter, the starting point for any visualization are the data. Since data can be stored in manifold ways, a visualization framework necessitates methods that allow a uniform handling of data from arbitrary data sources. It has already been indicated that the relational data model is widely used for managing and storing data. Thus, it is suggested to provide an interface for relationally structured data. Adhering to a relational data interface has the advantage of standardized data access based on tuples and attributes as well as data filtering based on selection and projection. A visualization framework should, therefore, support direct

access to relational database management systems (RDBMS) as well as to relational file formats (e.g., CSV[1]). The data interface of a visualization framework should be easy to extend to support further data sources. Another data-related issue is the dependency of data on time. To expand the range of applicability, a visualization framework should be able to cope with static as well as dynamic (i.e., time varying) data.

Visualizations Similar to supporting various data sources, providing different visualization techniques is also essential to support users in achieving their task at hand. The availability of diverse techniques gives users the option to choose the technique that suits their particular visualization problem best. From the system's point of view, the visualization techniques to be integrated into the visualization framework need to implement a common interface. This interface should adhere to the visualization pipeline and the parameterized visualization transformation (see Definition 2.2 on page 6). Visualization techniques that implement the common interface should provide a rich set of parameters to control the data analysis, the filtering, the mapping, and the rendering. Preferably, the visualization parameters should be interactively adjustable via direct manipulation as suggested by [Shneiderman, 1983], meaning that users can directly manipulate the visual representation. In addition to that, a rich set of visualization parameters opens up more options for intuitively perceivable event representations through automatic parameter adjustments.

A framework that provides diverse visualization techniques and handles different data sources should also provide multiple views. Multiple views are particularly useful to support one of the essential tasks of visualization – the visual comparison. Multiple views facilitate not only the comparison of different portions of a dataset (e.g., comparison of different time steps), but also comparisons across visualization techniques and dataset boundaries (e.g., to compare dataset A visualized by technique X with dataset B visualized by technique Y). Even views that have been generated by the same, but differently parameterized visualization technique can foster the understanding of the represented data. Therefore, supporting multiple views is mandatory for a visualization framework.

In the light of multiple views, coordination of these views becomes an issue. By coordination it is meant that changes applied to one view are automatically applied to all other views (that are coordinated). Especially when using multiple views for comparison purposes, it is important for users to have the option to coordinate the views participating in the comparison. This allows applying, for instance, a rotation to one view while all coordinated views are rotated as well. For an easy coordination of views, the underlying mechanisms should be as flexible as to enable users to choose freely which parameters of which views should be coordinated.

Events In addition to the aforementioned general requirements, an event-based visualization framework must consider the special needs of integrating events into visualization. The most obvious requirement is that an event-based visualization framework must be able to handle the flow of event types and event instances through the system.

[1] CSV = Comma Separated Values; a file format that is commonly used to store data tables.

This implies that the framework must be developed in close accordance with the model of event-based visualization, i.e., interfaces for event specification, event detection, and event representation must be provided. With respect to the event specification, it is important to provide adequate tools for users with different experience. Even though different event specification methods should be provided, they all should rely on a common interface. This enables distribution and reuse of event types among users and across system boundaries. The requirements regarding the event detection can be easily derived. The event detection should also be based on a common interface. Concrete implementations of this interface should consider the specific needs of the addressed abstract event type; especially the computational effort must be taken into account. To create visual representations adapted to the occurrence of relevant events, an event-based visualization framework must allow visualization techniques to access event instances and other event-related information (e.g., time of last event detection). This, in turn, enables the visualization techniques to trigger actions and/or processes to adapt the visualization parameters, for which a uniform access is desirable, or to prepare events for their explicit representation.

Users Self evidently, any visualization framework necessitates an adequate user interface. The intuitiveness of the user interface is even more important for an event-based visualization framework. Only if the user interface allows easy access to the available functionality, can users successfully apply event-based visualization to accomplish their data analysis tasks.

If a visualization framework meets all the requirements described up to now, it allows data from different sources to be represented in multiple, possibly coordinated views that are adapted to the occurrence of event instances of arbitrary event types. Because of this complexity, some form of user management is needed as well, meaning that an event-based visualization framework should be able to collect associations between users, datasets, visualization techniques, and relevant event types (see Section 4.4.2). These associations should then be used to drive a user interface that automatically suggests datasets to open, visualization techniques to use, and event types to consider depending on the current user. To support dynamic data, it is also necessary to store information about which portions of the available datasets have already been visited by particular users and which parts are new, i.e., could contain events that users are interested in. Additionally, previously successful visualization results and personal preferences should be handled in a user management component.

Requirements summary When developing the architecture of the framework eVis, it was the goal to meet the following key requirements:

- Enable uniform data access backed by the relational data model.
- Provide visualization techniques that adhere to the parameterized visualization transformation. Allow multiple views and coordination of these views.
- Compile the flow of event types and event instances into the framework architecture in accordance with the model of event-based visualization.

- Provide an adequate user interface to the framework's functionality backed by a user management component.
- Consider common visualization requirements.

Not every single requirement one can think of in the context of a visualization framework can be described here in detail. It is even difficult to gather a complete collection of visualization requirements from the current literature. The reason being that the analysis of requirements for successfully designing visualization systems is still a current topic of research. The interested reader is referred to [Tang et al., 2004], where several design choices for developing visualizations are explained. The authors discuss different aspects of visualization and provide valuable implementation guidelines. Specifically, data transformations, the modularization of data, visualization, and interaction, as well as graphics-related performance issues are considered. [Lange et al., 2006] also investigate factors influencing the visualization design and resulting implications. For a thorough analysis of visualization requirements in conjunction with concepts for (semi-) automatic meta-data-driven visualization design, readers are referred to [Nocke, 2007].

5.2. Framework Architecture

A prerequisite to a successful implementation of event-based visualization is a well-designed framework architecture. With the previously described requirements in mind, the architecture of the framework eVis has been developed. The visualization pipeline (see Section 2.1) and the model of event-based visualization (see Section 4.2) are integral parts of the architecture. A conceptual outline of the architecture of eVis and brief descriptions of its basic parts are given in this section.

Figure 5.1 depicts an overview of the architecture, in which the basic parts are *data import*, *event specification*, *management*, *event detection*, and *event-based visualization*. It can also be seen that the architecture makes use of interfaces, which are implemented by different modules, to provide different functionality and to be open for further extensions.

Figure 5.1 illustrates how data, event types, and event instances are processed through the different parts of the architecture. Basically, the starting point of the data flow is the data import, which provides facilities for reading datasets from different data sources and mapping them to an internal relational data structure, i.e., for preparing raw datasets for later event-based visualization. On the other hand, the event specification is the starting point of the event-related flow. The event specification provides the interface for incorporating event types into the framework. In the suggested architecture, data and event types are further processed through the management component. The management component is a top-level module that is responsible for managing all resources in the framework. Here, resources denote data sources, actually open datasets, associated event types, visualization views, user interface controls as well as users and associated preferences. Having been processed by the management component, data and event types are passed over to the event detection. Lastly, data and detected event instances are handed over to visualization techniques

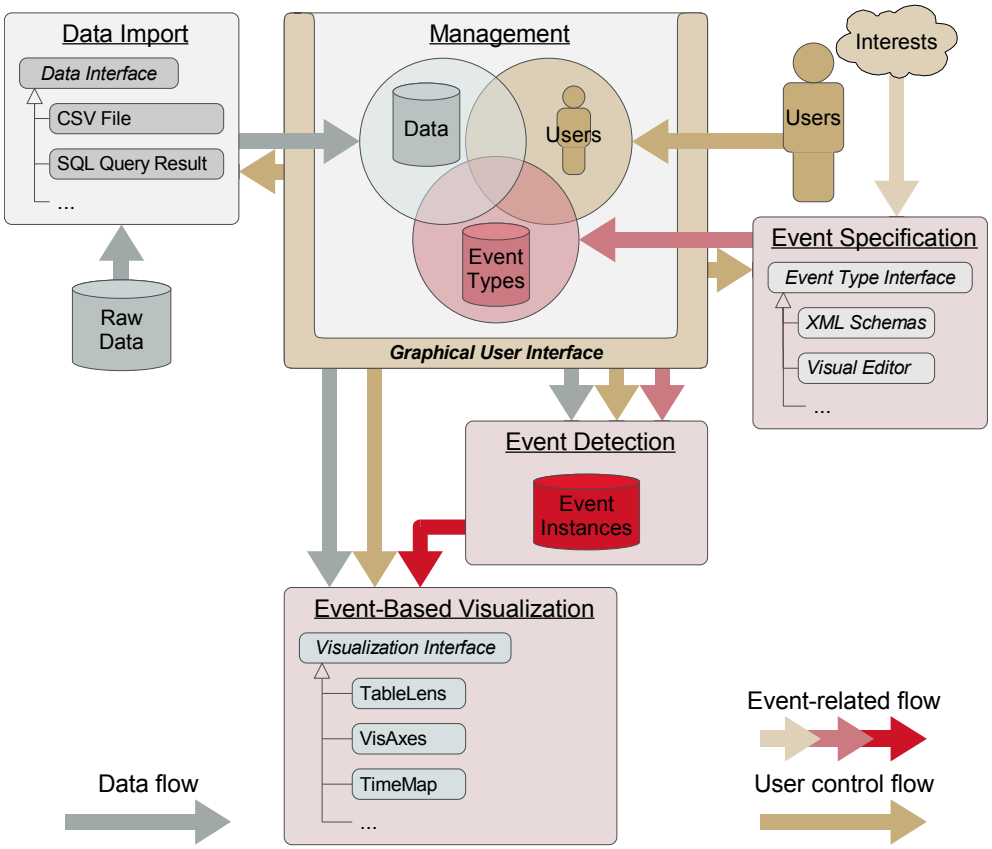


Figure 5.1.: Architecture of the event-based visualization framework eVis.

that either are enhanced with actions or processes, which adjust the visualization parameters according to the detected events, or are particularly suited to represent events explicitly.

Having provided an initial overview of the architecture of eVis, it is now possible to describe the basic parts as well as their tasks and functions. In addition to the following brief descriptions, the Sections 5.3 and 5.4.1 to 5.4.3 will provide more detail on those components involving visual methods, namely the event specification via visual editors and event-enhanced visualization techniques.

Data import The data import defines the interface between data sources and the event-based visualization framework. As mentioned earlier, the data interface provides uniform access to relational data structures, i.e., data that comprise attributes and tuples. To allow the visualization of a wide range of datasets, two implementations of the data interface provide access to remote relational database management sys-

tems (RDBMS) via SQL queries and to CSV files stored on a local file system. Both data import modules work asynchronously to keep the user interface responsive during loading operations.

The data interface also provides access to meta-data frequently required by visualization techniques (e.g., information about value ranges or derived statistics like minimum, maximum, average, mean). The data interface is capable of automatically maintaining this information for dynamic datasets. An essential task of the data interface is to inform the management component about changes to the data.

Event specification The event specification is also based on an interface that is common to all implementing modules. To allow easy exchange of event types, this interface uses XML grammar to describe event types. This has necessitated the development of XML schemas that cover the syntax of the introduced abstract event types. Until now, eVis supports the abstract tuple event type, the abstract attribute event type, and the abstract composite event type by providing appropriate XML schemas. Since XML is human readable, expert data analysts can directly specify event types using their favorite XML editing tool.

Nonetheless, it is important to provide means for supporting the event specification. Therefore, a visual interface to the event specification formalism has been developed. Its implementation as a visual editor plug-in for a general editor environment is described in detail in Section 5.3.

Management The management component is a top-level component that provides, via an easy to operate user interface, functionality to open and close visualization views, to load datasets from different data sources into a data pool, and to load various event types into an event type pool. The latter pools are the basis for further use of data and event types throughout the framework. Until now, users have to assign datasets and event types from the respective pools to visualization views manually. To facilitate this step, the management component preserves a list of recently visualized datasets and recently used event types. However, further automatic support is required to simplify this task.

Since a dataset may be visualized in multiple views, it is the task of the management component to notify the different views of changes to dynamic datasets and to trigger a re-detection of events. An event detection must also be conducted if a different dataset or further event types are assigned to a view.

To allow users to parameterize the visualization techniques differently for each view, the management component automatically generates a graphical user interface for the visualization parameters offered by the visualization techniques currently used. A common interface for visualization parameters has been developed to accomplish this task. Since it makes no sense to provide access to the parameters of all visual representations all at once, only the parameters of the currently focused[2] view are accessible via the user interface.

[2] Whether or not a view is focused is determined by the underlying windowing system (usually with respect to the mouse cursor position).

Event Detection As its name indicates, this module is responsible for detecting event instances. Since different abstract event types require different detection methods, a dedicated event detection mechanism must be implemented for each considered abstract event type. Modules for detecting tuple events, attribute events, and composite events have been implemented. These modules apply the basic event detection mechanisms described in Section 4.5 (except of mapping tuple event types to SQL queries, performance increasing methods have not yet been integrated). To avoid conducting the event detection separately for each view, in a first step all event types assigned to views representing the same dataset are collected from the event type pool. Secondly, a per dataset event detection is run. Each dataset is then associated with a list of detected event instances. From this list further composite events can be recursively derived and added to the same list. To allow for implicit and explicit event representations, the list of event instances is freely accessible for visualization techniques.

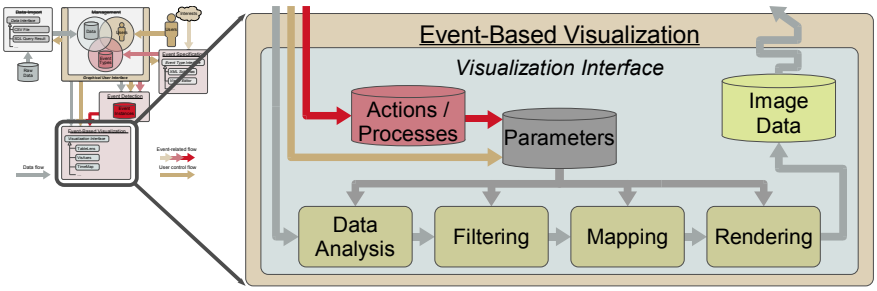


Figure 5.2.: The visualization interface of eVis.

Event-based visualization Similar to the previously described components, the visualization part of eVis is also based on a well defined interface. Basically, this interface adheres to the parameterized visualization transformation and the visualization pipeline. What is added in the light of event-based visualization are actions and processes, which can be used to control the parameters of the visualization. A schematic illustration of the event-based visualization interface is given in Figure 5.2. Any visualization technique that implements this interface can be used in the framework eVis.

Later in this thesis various visual representations are presented that have been incorporated into eVis, and hence can be used in an event-based manner. In particular, a simple TableLens, axes-based visualizations, and map-based representations are described in Sections 5.4.1, 5.4.2, and 5.4.3.

The previous descriptions provide a basic outline of the framework eVis. Since the focus of this thesis is on visual methods, the visual components of the framework eVis are subject to more detailed explanations. In the next section, visual aspects of the event specification will be discussed. Subsequently, it will be described how events are considered in different event-based visualization techniques.

5.3. Event Specification via Visual Editors

The event specification as described in Chapter 4 is based on predicate logic formulas (and some extensions to handle sequence event types). A user-centered event specification model has been developed that allows users with different experience to specify their interests as event types at different levels of abstraction (i.e., direct specification, specification by parameterization, and specification by selection; see Section 4.4.2). Although this model facilitates the event specification, there is still the need to provide support at the level of direct specification.

In recent years, visual editors have emerged in virtually all fields of Computer Science (e.g., visual programming environments or GUI designers). These editors provide interactive visual interfaces to models or languages that are mostly abstract, but not inherently of visual nature (e.g., query or programming languages). Experiences show that such visual support can significantly ease the use of any formalism on computer systems [Catarci et al., 1997]. Therefore, it was the goal to develop a visual editor that facilitates the specification of event types in an event-based visualization environment.

Requirements When developing interactive visual aids for the event specification, several requirements must be taken into account. An obvious requirement is that the tools to be developed have to facilitate the event specification process, but must not overwhelm the user with functionality that is difficult to operate. To achieve this requirement, first of all, the event specification formalism must be mapped to an intuitive visual model that can be represented and interactively edited on a computer display. In other words, every element used in the event specification formalism (i.e., the relational and logical notions used) must be associated with a visual counterpart that represents the element on the display. Accordingly, possible relations between the elements have to be considered.

However, since the model of event-based visualization is for general use and not limited to the abstract event types introduced so far (i.e., $\widehat{T}$, $\widehat{\mathcal{A}}$, $\widehat{S}$, and $\widehat{C}$), further abstract event types can possibly be added in the future. Therefore, the desired visual editor should have a modular structure that is easily extensible. This is also important when facing different application scenarios where it might be useful to provide adapted visual editors to emphasize the specifics of the application background. For instance, a user who has to analyze spatio-temporal census data can be provided with an editor that focuses on the use of spatio-temporal predicates, whereas in another editor these predicates can be even omitted for the sake of simplicity.

In conclusion, there is not the one and only visual editor for the event specification. Due to the diversity of abstract event types and application scenarios, editors are required that are adapted to the tasks at hand. Nonetheless, all editors should be embedded into a common runtime environment. The advantage of doing so is that a uniform appearance of all visual editors can be easily achieved. In addition to that, user interface components and interaction techniques are uniform and can be used equally in all developed editors. Aiming at uniformity implies not only less learning effort for the users of the visual editors, but also less implementation effort, because redundant implementations are avoided.

Implementation To enable the development of visual editors that meet the stated requirements, a common editor environment has been designed. The editor environment has been inspired by IBM's Eclipse Platform, which *"is an IDE for anything, and for nothing in particular."* Accordingly, the developed editor environment is based on a micro-kernel that provides only very basic functionality. In particular, the kernel merely drives a plug-in system that is responsible for linking any additional functionality (e.g., a concrete visual editor) into the editor environment.

To build a common basis for the later integration of arbitrary visual editor plug-ins, three fundamental components have been implemented (also as plug-ins):

- Project management,
- Window management, and
- Graphical editor interface.

The project management provides methods to handle the files involved in the event specification. It enables basic operations like reading and writing of event types stored in XML files, and grouping of files according to the users needs (e.g., to group event types that are related to a particular dataset or visualization task). The project management is also responsible for supporting the validation of the specified event types with respect to the characteristics of data stored in relational databases. To accomplish this, the project management queries available datasets and their relational schemas from the database server that stores the data under investigation. The retrieved meta-data is made available for other plug-ins.

Whereas the project management is a data-centric component, the window management incorporates first visual elements into the editor environment. Elementary views like tree views for the project management, tool boxes for the user interface, or abstract editor windows have been integrated in the window management plug-in. All views have been developed in accordance with the model-view-controller (MVC) pattern, which is an accepted reference model for user interface development [Gamma et al., 1994]. Since multiple windows (project view, property view, editor views) must be visible at the same time, special window layout algorithms have been incorporated (e.g., docking of windows).

Using the basic windowing and user interface components of the previously described plug-in, it is possible to load the abstract graphical editor interface into the kernel space. This interface provides the basis for all visual event specification editors. It defines in an abstract manner the relationships between elements of the event specification formalism and possible graphical representations of these elements and their relations with each other. Furthermore, mechanisms are provided that ensure that only valid event types are specified. Thanks to the formal foundation of the event specification, the syntactic correctness of currently edited event types can be validated easily, and users can be provided with immediate feedback about the event types' correctness. Additionally, basic tests of the semantic plausibility are conducted. For these tests, the data types of the attributes that are used in an event type are fetched from the meta-data provided by the project management plug-in. The data types are then tested for their applicability

with respect to the predicates and functions used in the event type (e.g., a categorical value must not be multiplied by a date). By the syntactic and semantic tests and the immediately provided feedback, users are significantly supported in specifying their interests as event types.

The previously described plug-ins are the basis for the development of concrete visual editors for the specification of event types. Besides a simple text editor, a visual editor for tuple event types has also been developed. The visual editor for tuple event types provides concrete visual counterparts of the elements that may appear in tuple event formulas (see Section 4.4): Formal elements are visually represented by small boxes that, in turn, may contain smaller boxes that represent parameters of the elements (e.g., operands). Figure 5.3 (a) depicts examples of such boxes. In addition to the element type, which is indicated by a small icon in the upper left corner of each elementary box, data types of operands are also indicated by small icons in internal boxes. To construct an event type, users can interactively connect the visual elements with each other. Connection can only be made if the resulting event type passes the validity and plausibility tests. Invalid connections are immediately indicated by changing the mouse cursor as illustrated in Figure 5.3 (b).

To illustrate the use of the visual editor for tuple event types, the event types introduced in the Examples 4.1, 4.2, and 4.3 (see page 56) have been redefined visually. The visual representations of these event types are depicted in Figure 5.4.

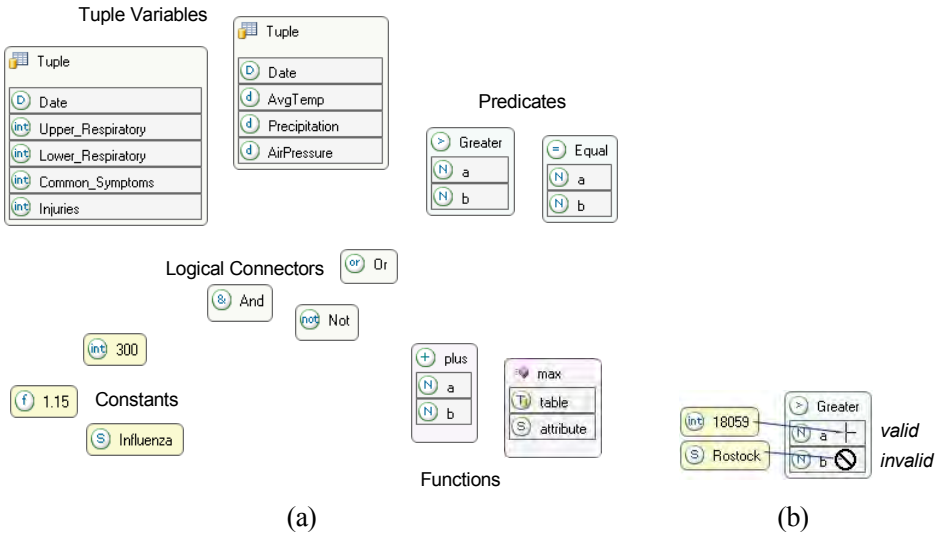


Figure 5.3.: Visual editor elements.
(a) A collection of visual elements available in the developed editor for tuple event types. (b) While specifying connections between these elements, users are provided with immediate feedback about the validity of the intended connection. In this case, the lower connection is invalid, since the operand requires a numeric connector (N), which "Rostock" is not (S).

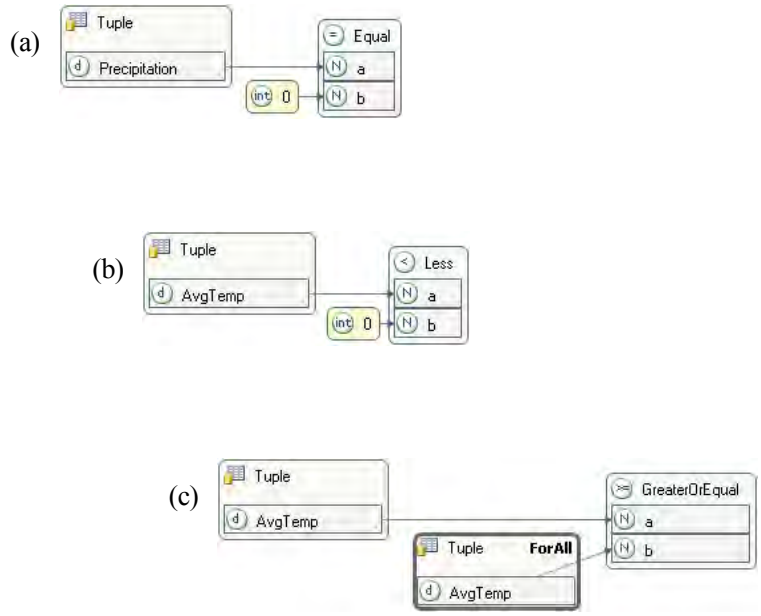


Figure 5.4.: Examples of visually specified event types.
The figure depicts visual representations of three event types that have been introduced as examples on page 56. (a) Example 4.1: days without precipitation; (b) Example 4.2: frosty days; (c) Example 4.3: maximum temperature.

In this section, an editor environment has been described that serves as a basis for the development of concrete visual editors to support users in specifying their interests as event types. The editor environment is based on a micro-kernel that can be extended via a plug-in mechanism. To demonstrate the benefits of visual editors, an editor for tuple event types has been implemented as such a plug-in[3]. This visual editor makes it easier for users to understand how the event specification formalism can be used to describe interesting event types with respect to the data under consideration.

Although the developed visual editor provides initial aid, there is still potential for more support of the event specification. Since the fundamental components are available, it should be easy to integrate further event type editors in the future. Especially the specification of sequence event types should be addressed by a dedicated visual editor. An interesting idea for future work is to investigate whether the ArcDiagrams invented in [Wattenberg, 2002] could be extended such that they can be used to define sequence event types.

[3]More details on the editor environment and the editor itself can be found in [Löffler and Willert, 2006]

5.4. Event-Based Visualizations

The goal of this section is to illustrate the use of visualization techniques that have been enhanced with event-based mechanisms. To achieve this goal, three different types of visual representations will be explained:

1. TableLens with implicit event representation,
2. Axes-based techniques with implicit event representation, and
3. Map-based techniques with explicit event representation.

At first, it is described how a simple TableLens representation can be improved by considering events implicitly via parameter adaptation (see Section 5.4.1). Subsequently, more complex visualizations that have been generated with the axes-based toolkit *VisAxes* are introduced. These techniques also incorporate diverse implicit event representations (see Section 5.4.2). Finally, it is explained how events can be coupled with the map-based toolkit *TimeMap*. In contrast to the former approaches, the focus is set on the explicit representation of events (see Section 5.4.2).

All three visualization approaches are explained in accordance with the following procedure: First, the general concern and the basic visual representations are described. Based on that, the event-related enhancements of each approach will be introduced.

5.4.1. An Event-Enhanced TableLens

The first simple example of a visualization technique that considers events is based on tables. A table can be used as a natural representation of a relational dataset. Many office software products make use of tabular spreadsheets. However, datasets with a large number of tuples cannot be displayed at a glance using common spreadsheets. Therefore, the TableLens has been developed [Rao and Card, 1994]. The TableLens reduces the required amount of display space by representing data values graphically as small bars, rather than printing them alphanumerically. Only for user-selected rows of interest, are data values printed directly into the table cells. This makes the TableLens an excellent tool for analyzing relational data.

To demonstrate the benefits of event-based visualization, a simple TableLens has been integrated into the framework eVis. The TableLens implementation used here differs slightly from the originally proposed one. Analogous to the original, small bars are used to represent data values per table cell. Whereas the original TableLens allows users to define focus rows, which are shown as common table cells, the TableLens implemented in eVis determines which rows are focused depending on a relative focus parameter. This parameter is equal to 0 if the first tuple of a dataset is focused, and is 1 if the last tuple is of interest. Depending on the focus parameter, a nonlinear 1D fisheye transformation [Keahey and Robertson, 1996] is applied to calculate positions and heights of the table rows. Having calculated the row heights, it is possible to associate each row with a separate font size that is used for printing data values. If the

font size drops below a certain threshold, no values will be printed, but instead will be represented by small bars. Figure 5.5 shows the implemented TableLens.

For the explanation of how events are incorporated in the TableLens, it is assumed that the TableLens is applied for visualizing a dynamic temporal health-related dataset. A variety of event types can be imagined that describe special situations of interest with respect to such data. Here, a tuple event type is assumed that is detected if a threshold has been exceeded. According to the suggested interpretations of tuple events a threshold event can be interpreted as a point of interest in the space spanned by the dataset's attributes (see Figure 4.11 (c) in Section 4.6.1). It is a natural requirement that these special points (i.e., rows) are highlighted in the TableLens. The focus parameter of the TableLens can be adjusted to accomplish an automatic highlighting according to detected events. For doing so, an action has been integrated into the TableLens. This action automatically sets the focus parameter according to the latest[4] detected tuple event in the table. Although this action performs a quite simple parameter adjustment, it yet achieves an automatic highlighting of the special situation. The user does not need to adjust the TableLens manually to view the tuples that have been identified as interesting. Figure 5.5 illustrates the difference between considering and not considering events in a TableLens representation.

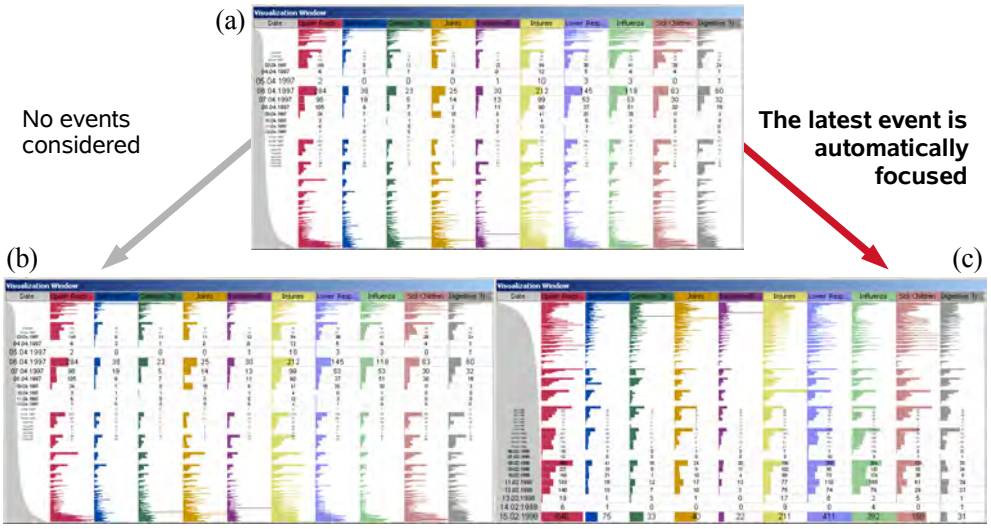


Figure 5.5.: An event-enhanced TableLens.
It is assumed that the TableLens has been presented as in (a) and that the event "number of cases of influenza is greater than 300" has been detected after new data were added to the dataset. (b) If the event is not considered, the user is not aware of the special situation. (c) With the help of event-based visualization it is possible to automatically focus on the event (in the last row of the TableLens); users can identify relevant information easily.

[4] An ordering of the tuples can be presumed for the considered dynamic temporal dataset.

For demonstration purposes, the automatic parameter adjustment has been implemented as an action. Alternatively, it would also make sense to realize the adjustment as a gradual change, i.e., as a process. In this case, the focus parameter would be changed over time from its current value (either set interactively or by a previous execution of the process) to the new value (determined by the last detected event). A reasonable duration for the animation lies between 0,5 and 2,0 seconds.

The TableLens has potential for further extensions. In the light of attribute events it would make sense to use a 2D fisheye transformation like the rectangular fisheye view as described in [Rauschenbach et al., 2000] to allow focusing on attributes (i.e., columns) as well. Additionally, fisheye transformations that support multiple foci are required to enable the highlighting of more than one event instance. Although these extensions are future work, the example of a TableLens that has been extended with an action to automatically adjust the focus parameter gives a first indication of what is possible when incorporating visualizations into the event-based framework eVis.

5.4.2. Axes-Based Visualizations with Event Integration

Visualization techniques that are based on visual axes have been used for a long time to represent data. Prominent examples are time diagrams, where two perpendicular axes are combined with a line plot to represent an attribute with respect to time. Another example of axes-based visualizations are Parallel Coordinates [Inselberg, 1998]. In this technique, multiple parallel axes are used to represent multivariate data, i.e., datasets with multiple attributes.

In the following, the visualization toolkit *VisAxes* is introduced. Besides allowing an easy implementation of classic axes-based visualizations, the VisAxes toolkit facilitates the development of novel axes-based visualization techniques. The novelty of the techniques presented next lies not only in the different arrangement of axes, but also in their capability to react to events detected in the data. In contrast to the previously described TableLens, the axes-based visualizations proposed in the following provide not only one, but several visualization parameters for automatic adjustment via actions or processes (implicit event representation).

Parts of the following sections have been published in [Tominski et al., 2003a], [Tominski et al., 2004], [Tominski and Schumann, 2004], and [Tominski et al., 2005a].

General Concept

The basic idea of axes-based visualization is to use visual axes to represent attributes of a dataset. When designing VisAxes, the major goal was to create a general toolkit that allows easy implementations of various axes-based visualizations. To achieve this, it is necessary to distinguish three conceptually different aspects:

1. Axis design,
2. Axes arrangement, and
3. Data representation.

The axis design determines how the value ranges of attributes are mapped onto visual axes. In turn, the axes arrangement describes how visual axes are arranged on the display. Finally, the data representation is responsible for actually visualizing data values of a dataset. The order of mentioning these aspects corresponds to the general process of computing axes-based visualizations. In a first step, data attributes (i.e., their value ranges) are assigned to axes that have certain axis designs. Secondly, the axes are arranged on the screen, and finally, data values are drawn (e.g., as dots). Usually, axes arrangement and data representation are strongly related. This means that axes arrangements impose restrictions on what data representations can be used. For a perpendicular arrangement of two axes, it is, for instance, possible to represent actual data values using line plots (e.g., time diagram), dots (e.g., Scatter Plot), bars (e.g., bar chart), and so forth. Other axes arrangements could require different data representations. When developing axes-based visualization techniques, it is, therefore, necessary to ascertain which data representations are compatible with particular axes arrangement.

The following paragraphs provide detailed descriptions of the axis designs and axes arrangements (as well as data representation) that have been incorporated into VisAxes. Possible enhancements for axis designs and axes arrangements are suggested subsequently.

Axis design In axes-based visualizations, each axis is associated with one attribute of the dataset to be visualized. Usually, each axis is scaled according to the minimum and maximum of the associated attribute's value range. If attributes have equal or similar value ranges, it is also possible to map more than one attribute onto the same axis. The design and the scale of an axis strongly depend on the type of data (i.e., nominal, ordinal, discrete, or continuous data) that is mapped onto the axis. Whereas quantitative (i.e., discrete or continuous) data can be easily mapped to axes by applying Euclidean distance measures, this is not possible for qualitative (i.e., nominal or ordinal) data, since distances between data values are not given explicitly. In the latter case, special methods (e.g., [Rosario et al., 2004]) must be used to enable the mapping of qualitative data to visual axes. Furthermore, if an attribute has a larger range of values, some form of value grouping is useful. To address this issue, hierarchical clustering can be applied. In this case or if an attribute is inherently hierarchically structured (e.g., the attribute time), it is natural to make use of the hierarchy when browsing the data. Another desirable feature is to have the ability to selectively map only certain user specified or automatically derived subranges of values onto the axes.

To meet these requirements and to provide a high degree of interaction with relatively large data sets, three different designs of interactive axes have been developed:

- Scroll axis,
- Focus+context axis, and
- Hierarchical axis.

A scroll axis is useful to represent a subrange of interest of a larger value range. The subrange can be defined interactively using a slider provided with a scroll axis. This slider can be widened or narrowed to expand or contract the subrange of interest. The position of the slider on the axis represents the position of the shown subrange with respect to the whole value range. Since the slider's position and extension are exported as visualization parameters, the represented subrange of interest can also be adjusted via actions or processes as a response to the occurrence of an event. As such, the scroll axis can be used to interactively or automatically zoom into certain subranges of interest or provide an overview of the whole value range of an attribute.

The second axis follows the focus+context approach. The focus+context axis is particularly useful in cases where it is necessary to show the whole value range of an attribute and at the same time to emphasize a particular data value. This is achieved by applying a 1D fisheye transformation [Keahey and Robertson, 1996] when mapping data values to the focus+context axis. This transformation is parameterized by the position of the focal point and the amount of magnification. Again these parameters can be adjusted either interactively using a special slider provided with the axis, or automatically using actions or processes. Especially the ability of the focus+context axis to emphasize a value of interest makes it useful for automatic adaptations on occurrence of events.

Inherently hierarchically structured attributes like time as well as attributes that have been hierarchically clustered can be represented using the hierarchical axis. This axis is subdivided into segments representing nodes of the hierarchy tree. To allow users to navigate within the hierarchy tree, the segments can be expanded and collapsed

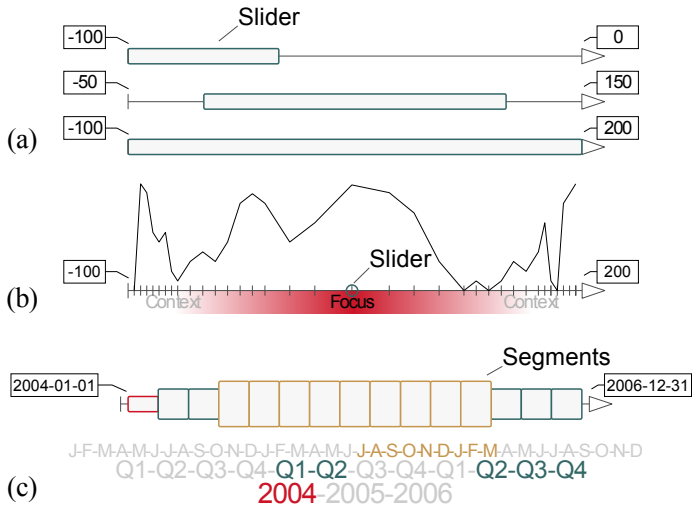


Figure 5.6.: Different types of visual axes.
(a) Scroll axes with different slider positions and widths; (b) A focus+context axis with focus and context regions; (c) A hierarchical axis representing the attribute time.

either interactively or automatically. This allows zooming (on a semantic level) into a certain subrange of interest while an aggregated view of the remaining value range is preserved at all times.

All three axis designs are depicted in Figure 5.6. The figure illustrates how the proposed designs map value ranges differently onto visual axes. All axes are available in VisAxes for creating axes arrangements as described next.

Axes arrangement The second aspect of axes-based visualization is the axes arrangement. It describes how the axes participating in the visualization are arranged on the display. Classic time diagrams, for instance, are made up of two axes that are arranged perpendicular to each other; the Parallel Coordinates technique implements an arrangement of parallel axes. Since it is mainly the axes arrangement that determines the effectiveness of axes-based visualizations, novel axes arrangements must be developed with care. To create effective arrangements, certain characteristics of the considered data must be taken into account. Here it makes sense to distinguish between attributes that span the frame of reference (independent attributes) and the attributes that have been measured in the frame of reference (dependent attributes). Certainly, for time-related data as considered in this thesis, the attribute time is an independent attribute, other attributes are dependent on time. When developing new axes arrangements, it is also important to achieve scalability of the arrangements. This means that axes arrangements should allow the visualization of datasets with different numbers of attributes. This demand is particularly important when considering automatically applied projection operations as response to the occurrence of some events. To allow for a high degree of interactivity and to enable automatic adjustments of axes arrangements with respect to occurred events, flexibility in terms of adjustability via parameters is also an important requirement.

Several novel 2D and 3D axes arrangements that satisfy these requirements have been incorporated into the toolkit VisAxes. In the next paragraphs, detailed descriptions of the most promising axes arrangements will given. These arrangements are:

- The 2D and 3D TimeWheel,
- The 2D and 3D MultiComb, and
- The 3D KiviatTube.

The TimeWheel is an axes arrangement that is particularly useful for visualizing time-related datasets in which time is the only independent attribute. The design rationale behind the TimeWheel is to emphasize on the extraordinary role of time. Therefore, this axes arrangement comprises a central, horizontally aligned reference axis (representing time) around which several attribute axes are arranged radially (representing dependent attributes). For the 2D version, all axes are arranged on the 2D plane (see Figure 5.7 (a)). For the 3D TimeWheel, the attribute axes are positioned on planes emanating radially from the common time axis in the 3D space (see Figure 5.7 (b)). To represent the data values in a TimeWheel, lines descent from the time axis to each of the attribute axes. Clearly speaking, for each time step, individual

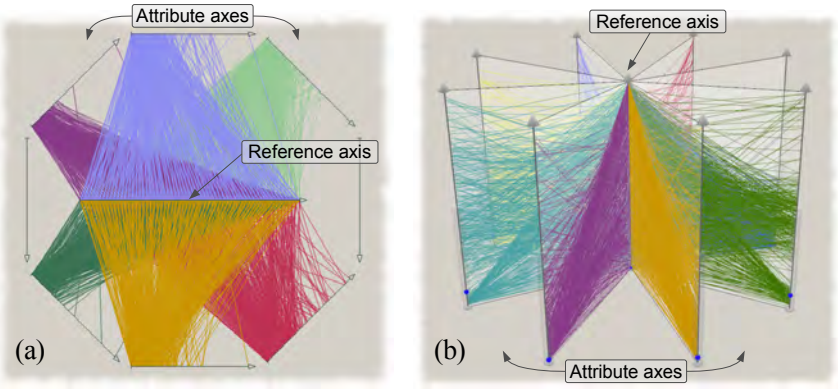


Figure 5.7.: The TimeWheel.
The figure shows a 2D TimeWheel (a) and a 3D TimeWheel (b). In the center of each TimeWheel, a reference axis depicts the attribute time. Around this central axis several attribute axes are arranged radially (and parallel with respect to the reference axis in 3D) representing time-dependent attributes. Data for each time step is visualized by lines descending from the reference axis to each of the attribute axes. In the 3D case, data wings are used to facilitate the perception of the lines.

line segments establish direct visual connections between time and all corresponding time-dependent values ($1 : n$). This type of data representation differs slightly from the one applied in Parallel Coordinates, where the line segments are drawn between neighboring axes only ($1 : 1$). This difference in the way of drawing the data, gives Parallel Coordinates the advantage that correlations between dependent variables could be revealed, which is barely possible when using the TimeWheel. On the other hand, when using Parallel Coordinates, it is difficult to identify the line segments that represent a particular time step. In contrast to that, the TimeWheel enables users to easily discern for all time-dependent attributes which data value occurred with respect to a particular time step. In turn, the TimeWheel can also be used to determine where in time certain data values were measured.

To enable an efficient use of the 3D TimeWheel, the issue of perception in a 3D presentation space must be addressed. In fact, position and orientation of line segments are not easy to perceive in a 3D space. Since position and orientation of a plane can be more easily interpreted, semi-transparent *data wings* are rendered between axes. The data wings facilitate the perception of which line segments are connected to which axes. By using alpha-blending when rendering the data wings, occlusion of information is reduced. The data wings are an important enhancement for axes-based visualizations in a 3D presentation space and, therefore, have been integrated in all developed 3D axes arrangements.

Although the TimeWheel is a promising technique for visualizing time-related data, it certainly requires some effort to understand the generated visual representations. On the other hand, line plots can be considered a classic visualization technique for time-

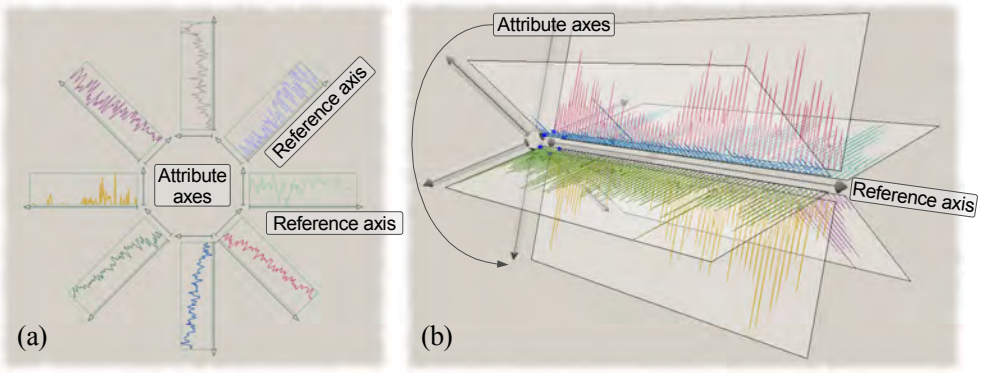


Figure 5.8.: The MultiComb.
The 2D MultiComb (a) uses line plots that are arranged radially on the display. Each plot represents the dependency of an attribute on the reference attribute (usually time). The 3D MultiComb (b) arranges plots radially around a common reference axis. Data wings are used to ease the perception of the plots.

related data; virtually every user is familiar with plot-based visual representations. However, simple line plots are limited in the number of time-dependent attributes that can be visualized. Therefore, the MultiComb arrangement[5], which is based on line plots, has been developed. The aim when developing the 2D and 3D versions of the MultiComb was to exploit the ease of use of plot-based representations and to overcome the limitations of simple line plots. Similar to the TimeWheel, the 2D MultiComb uses a radial arrangement of axes. In this case, pairs of perpendicular axes are arranged. Each pair of axes constitutes a separate time diagram in which one axis is used to encode time and the second axis encodes a time-dependent attribute. Data values are represented by drawing plots for each pair of axes. Two possibilities do exist for arranging pairs of axes as a 2D MultiComb. The first option is an arrangement in which the reference axes, and hence the plots extend outward from the center of the 2D MultiComb (see Figure 5.8 (a)). Alternatively, axes representing the dependent attributes and axes representing time can be flipped, resulting in the plots appearing circular (see Figure 5.14 (a) on page 126).

For the 3D MultiComb, all plots share one common reference axis representing time. Time-dependent attributes are mapped to attribute axes that are perpendicular to the reference axis. With respect to the common reference axis, each of the attribute axes spans a plane extending at regularly spaced angles in the 3D space (see Figure 5.8 (b)). The respective line plots are drawn in these planes. Using the 2D or the 3D MultiComb, it is possible to visualize multiple time-dependent attributes with respect to time. Since both axes arrangements rely on simple line plots, they are intuitively understandable.

[5]The MultiComb was inspired by [Abello and Korn, 2002].

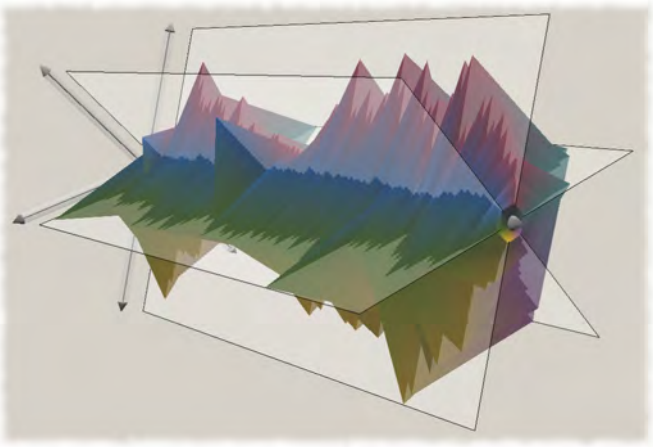


Figure 5.9.: The 3D KiviatTube.
The 3D KiviatTube uses the same arrangement of axes as the 3D MultiComb. However, in the 3D KiviatTube, data are not represented as line plots, but as a closed three-dimensional surface. This supports the understanding of relations between the represented attributes.

The third axes-based visualization technique to be introduced is the 3D KiviatTube. Basically, the 3D KiviatTube uses the same arrangement of axes as the 3D MultiComb, i.e., axes representing time-dependent attributes share in a right angle one common reference axis representing time (see Figure 5.9). The difference here is that not plots are drawn, but the whole dataset is represented as a closed surface (which is not possible in 2D). This surface is created by connecting points from neighboring plots and successive time steps to triangles. By doing this for all time steps, it is possible to render the whole dataset such that it appears as a closed surface. To put it differently, one can imagine a common Kiviat Diagram representing several time-dependent attributes for multiple time steps (see Figure 5.10 (f)). Similar to expanding an accordion, the 3D KiviatTube is generated by extruding the 2D Kiviat Diagram along a time axis that extends perpendicular to the original Kiviat Diagram. As such, the 3D KiviatTube encodes not only the evolution of attributes over time, but also correlations among attributes. However, when using the 3D KiviatTube, it can be difficult to identify narrow valleys in the surface of the tube due to 3D occlusion. The functionalities offered by the described interactive axes are useful to alleviate this problem; common 3D interactions are indispensable.

As indicated, VisAxes has been designed as a toolkit for experimenting with various arrangements of axes. In the previous discussion, the promising techniques TimeWheel, MultiComb, and KiviatTube have been described. In addition to that, classic techniques like Chart Stack, Scatter Plot Matrix, Parallel Coordinates, and Kiviat Diagram have been implemented. Currently, further axes arrangements are undergoing evaluation of their usefulness for multivariate data visualization. A collection of axes arrangements generated with the VisAxes toolkit is presented in Figure 5.10.

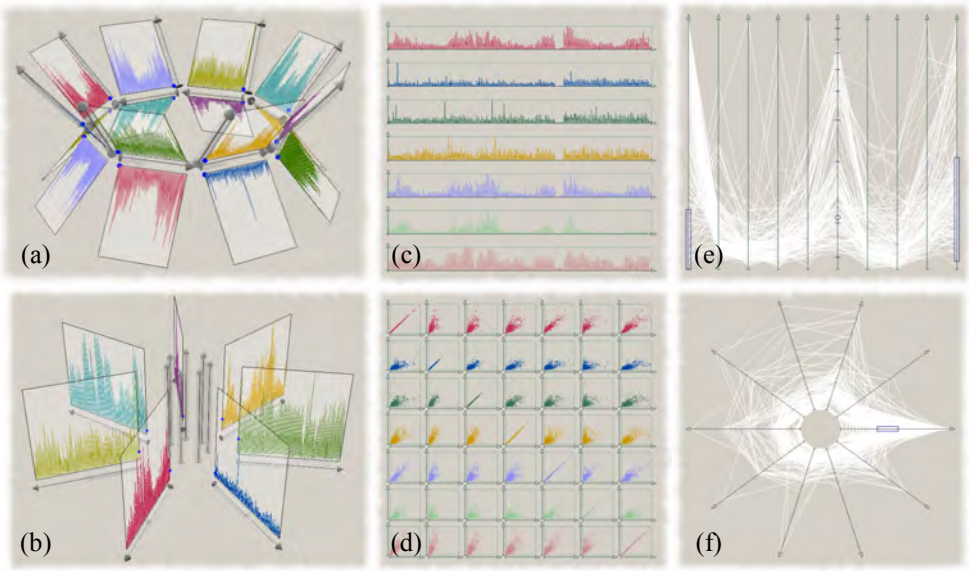


Figure 5.10.: Axes-based visualization created with the VisAxes toolkit.
(a) 3D Double Plot; (b) Alternative 3D MultiComb; (c) Chart Stack; (d) Scatter Plot Matrix; (e) Parallel Coordinates; (f) Kiviat Diagram.

Additional Enhancements The explained visualization techniques can be improved in several ways. In the following it is described how axes-based visualizations in general and specific axes arrangements in particular can be enhanced by providing additional interaction and visualization possibilities. The proposed enhancements aim at improving the expressiveness and increasing the applicability of axes-based visual representations for purposes of event-based visualization.

General extensions A major issue in VisAxes is interaction. Therefore, all axes arrangements have been designed to allow as much interaction as required for efficiently using the proposed techniques. An important interaction requirement is the reordering of axes. In VisAxes, this is realized by manipulating the attribute-axis-association using a dedicated user interface (see Figure 5.11). This interface allows not only adding and removing of attributes, but also reordering of attributes already participating in the axes arrangement.

For arrangements that compute a radial layout of axes, an additional visualization parameter allows users to rotate the axes arrangement interactively. This is important, since attributes represented at different angles in a radial arrangement are perceived differently. For the TimeWheel, for instance, the attribute axes can be analyzed best if they are arranged parallel with respect to the reference axis (see Figure 5.12 (a)). The analysis becomes more difficult if the angle between an attribute axis and the reference axis increases. To allow users to bring the

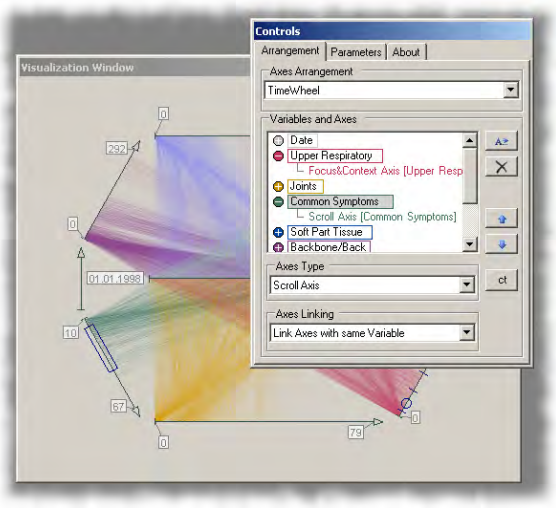


Figure 5.11.: User interface for managing visualized attributes.
The user interface allows adding, removing, and reordering of axes. It is also possible to change the type of axes and the method for linking axis interactions.

attributes they are interested in to the dedicated analysis position, the possibility to rotate the attribute axes around the central reference axis is crucial for the TimeWheel. The MultiComb requires interactive rotation as well. For the MultiComb, however, rotation is used to bring plots to a position such that the axes of the plots are aligned with the x-axis and the y-axis of the display, respectively (see Figure 5.12 (b)).

A further general extension addresses visualizations in which the representations of data items interfere. This is the case not only for arrangements that use line segments to represent data items (e.g., TimeWheel, Parallel Coordinates), but also for Scatter Plots and Scatter Plot Matrices. In these cases, it makes sense to utilize alpha-blending when drawing data items. By interactively adjusting the translucency of the data representation, users can control how much they can "see through" overlapping data items (see Figure 5.12 (c)). This also enables the detection of dense data regions.

Labeling When using axes-based visualizations, it is important to provide users with information about the represented attributes and respective value ranges. This can be achieved by labeling, i.e., annotating each axis with additional information (e.g., attribute name, minimum and maximum value). However, labeling arbitrarily arranged axes is a challenging task. On the one hand, additional information must be integrated into the visual representation; on the other hand, occlusion of information in the visual representation must be avoided. VisAxes implements a twofold strategy to achieve these requirements (see Figure 5.12 (c)). First,

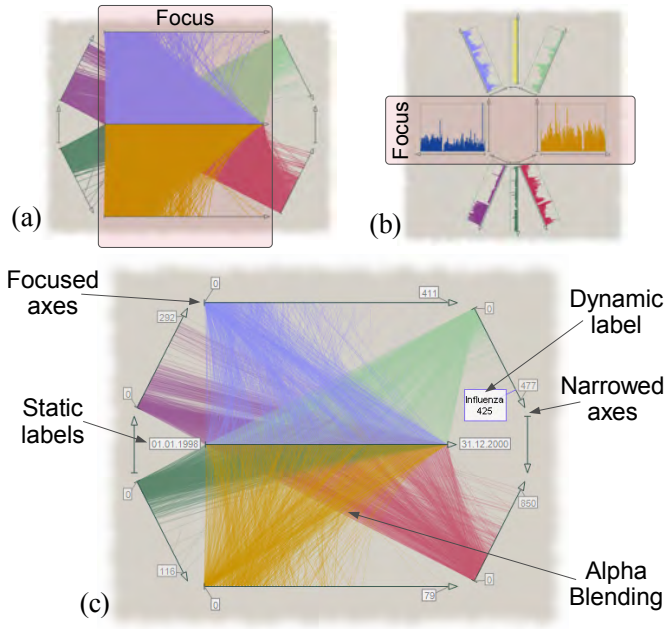


Figure 5.12.: Enhancements for axes-based visualizations.
(a) Focus region of the 2D TimeWheel; (b) Focus region of the 2D MultiComb; (c) Enhanced 2D TimeWheel with static and dynamic labeling, alpha-blending, and focused axes.

each axes arrangement implements its own labeling algorithm. Due to the tight coupling to a particular arrangement the algorithms are less general, and thus easier to implement. This first possibility provides a static minimum/maximum labeling that avoids occlusion of information. The second part of the labeling strategy is independent of the axes arrangement. Here, only on demand (e.g., if the user presses a mouse button) a single label is drawn with respect to the current position of the mouse cursor. To determine what information should be provided in the dynamic label, a perpendicular is dropped from the mouse cursor to the nearest axis. The label shows the name of the attribute that is associated with the nearest axis, and also the concrete value at the position where the perpendicular intersects the axis. Since this second labeling possibility has an on demand character, special procedures for avoiding occlusion of information are not considered. Nonetheless, both labeling methods utilize alpha-blending to minimize side effects.

Markers Similar to labels, markers can provide additional information in axes-based visualizations. In VisAxes, markers are small geometric primitives (e.g., arrows) that can be attached to visual axes (see Figure 5.14). Markers are associated with a specific value they represent. According to this value, markers are automatically positioned at axes. This makes markers useful for conveying aggregated values

like averages, means, or quartiles. In conjunction with scroll axes representing time, markers can also represent values aggregated with respect to value ranges preceding ("past") or succeeding ("future") the current subrange of interest. This allows users to analyze a subrange of interest in detail while, for instance, average values of "past" and "future" are preserved at all times.

Focus+context enhancements As indicated earlier in this thesis, focus+context approaches can be used to cope with large datasets. VisAxes incorporates various methods that rely on the focus+context concept. Besides the focus+context axis, which supports datasets that are large in terms of the number of tuples, further methods have been developed that facilitate the visualization of datasets that are large in terms of the number of attributes.

For 2D axes arrangements that use radial axes layouts (i.e., the TimeWheel and the MultiComb), the focus+context concept has been integrated directly into the calculation of the layout. Whereas in the original implementation all axes of a radial arrangement are provided with the same amount of display space, integrating focus+context aims at providing axes at exposed positions with more display space (focus) and reducing the space used for less-relevant attributes (context). The focus and context regions of the TimeWheel and the MultiComb are depicted in Figure 5.12 (a) and (b), respectively. The focus regions mark areas in which the represented attributes can be analyzed best. Knowing these areas, it is possible to provide axes being rotated toward focus regions with gradually increasing display space. Axes farther away from the focus regions (i.e., close to the context regions) are narrowed accordingly. To further reduce the visual presence of information displayed in the context regions, color-fading is applied when axes are rotated toward the context regions such that the representation of data items disappears gradually (see Figure 5.13 (a)).

For the 3D TimeWheel and the 3D MultiComb, a special focus+context interaction has been developed. The "folder style" interaction[6] allows users to browse attributes represented in a 3D arrangement similar to opening a folder (or an address register). When applying the "folder style" two axes are focused and can be easily compared; the other (context) axes are still visible in the background (see Figure 5.13 (b)).

Linking The extension described next regards the interaction facilities provided by the proposed axis designs. So far, user interactions are exclusively applied to the axis being the source of the interaction. This way of considering user interactions is sufficient for axes arrangements that assign a particular attribute to no more than one axis (e.g., TimeWheel or Parallel Coordinates). In cases where an attribute is mapped to multiple axes (e.g., attribute time in the MultiComb or attributes in Scatter Plot Matrices), it would be better to allow for interactions in a linked fashion. Similar to the coordination of views, this means that when the user browses through an attribute's value range using the scroll axis, all

[6]The "folder style" interaction was inspired by Anton Fuhrmann.

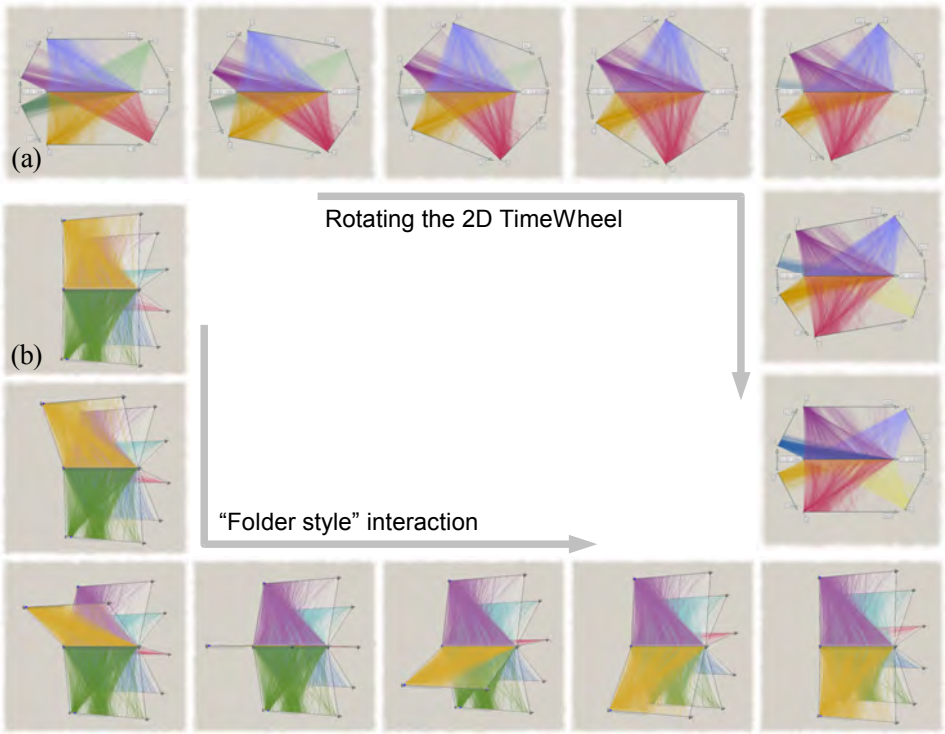


Figure 5.13.: Focus+context interaction in the TimeWheel.
(a) A series of screen shots taken while rotating the 2D TimeWheel. The axes are scaled according to their position with respect to the focus region as depicted in Figure 5.12 (a). Additionally, color-fading is applied. (b) A series of frames demonstrating the "folder style" interaction integrated in the 3D TimeWheel. The axes are "opened up" like a folder.

other axes representing the same attribute should automatically be scrolled as well. Accordingly, focusing on a certain value of an attribute (e.g., by using the focus+context axis) should result in focusing on the same value at all axes representing that attribute. Consequently, a flexible linking mechanism has been developed. This mechanism provides two ways of matching linked axes. On the one hand, an absolute linking is possible, meaning that exact values are matched. This is useful for linking axes that represent the same attribute. On the other hand, a relative linking has been implemented. For this case, not exact values are matched, but their relative positions within the attributes' value ranges. This allows linking of axes that do not necessarily need to represent the same attribute. Relative linking is particularly useful in combination with scroll axes to accomplish tasks like "Show all 'high' values!". In VisAxes, users can freely choose if and what kind of linking they prefer for their particular task at hand (see Figure 5.11).

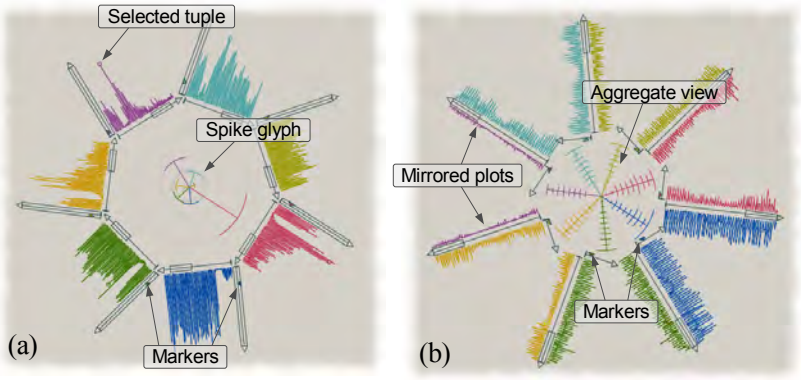


Figure 5.14.: Additional views for the 2D MultiComb.
(a) A spike glyph is positioned in the center of a 2D MultiComb to depict all attribute values for a selected time step. The spike glyph supports the direct comparison of attribute values for the selected time step. (b) Alternatively, an aggregate view can be used to visualize the "history" of browsed time steps in an aggregated fashion. To facilitate comparison of attributes, also mirrored plots can be added to the display.

Additional views The last enhancement to be described solely addresses the MultiComb. The implementation of the MultiComb leaves unused display space in the center of the axes-arrangement. This free space can be utilized to present additional information via two alternative views. The first view is a spike glyph (see Figure 5.14 (a)) that represents a single data tuple that has been selected by the user. In a spike glyph, the lengths of the spikes encode tuple values. The spike glyph is particularly useful for comparison tasks. A second additional view is the aggregate view (see Figure 5.14 (b)). This view can only be applied in conjunction with scroll axes representing the independent attribute (usually time). In this case, the range of values preceding the subrange of interest currently displayed at the scroll axes can be considered the "data history". In the aggregate view, this history is represented in a compact fashion using aggregated values. For creating the aggregate view, a certain number of aggregated values are computed (10 in the example of Figure 5.14 (b)). Each of these aggregated values represents the average of a user-defined number of original data values. The computed average values are encoded to the angles of small arcs being aligned at additional axes extending from the center of the display. By looking at the aggregate view[7] while browsing through time, users still get an idea of how the data looked like a few interaction steps ago. To further facilitate the visual analysis, mirrored plots, which support users in comparing the evolution of data values, are available on demand (see Figure 5.14 (b)).

[7] A similar representation is used in [Havre et al., 2001] to represent search results.

With these extensions the axes-based visualization techniques described earlier become valuable interactive tools for the analysis of multivariate data in general and multivariate time series in particular. Since the toolkit VisAxes has been designed to export visualization parameters where required, the developed axes-based visualizations are subject to automatic adjustments according to the occurrence of events. Such adjustments are described in the next section.

Integration of Events

In the previous paragraphs, several axes-based visualizations have been described. These techniques provide diverse visualization parameters for interactively adapting the visual representation. This section will explain the implicit event representations (i.e., automatic parameter adjustments) that have been integrated into the proposed axes-based visualization techniques. The examples demonstrate the integration of events with respect to the different event interpretations described in Section 4.6.1.

Interesting data values or data points First, events are considered that can be interpreted as interesting data values or points of interest in the space that is spanned by the attributes of a dataset (see Figure 4.11 (b), (c), (d), and (e) on page 78). An easy to interpret representation of such event instances can be accomplished by using the markers available in VisAxes. As already described, a marker is parameterized by a representative value according to which markers are automatically positioned at axes. This allows an emphasis on events via actions that adjust marker values with respect to detected values of interest. Since markers can be of different shape and color, it is possible to communicate the type of event instances as well (up to a certain number of different event types). This makes markers a simple, yet very useful mechanism for representing events. Besides using markers, it is also possible to take advantage of the interactive axes available in VisAxes. To represent events interpreted as interesting data values, the focus+context axis is the axis design of choice. Its inherent possibility to focus on a particular value within an attribute's value range allows implementing actions (or processes alternatively) that automatically focus on data values of interest. Depending on the axis designs currently used in a visualization session, it might be necessary to switch the axis design by setting the corresponding visualization parameter. Although the focus+context axis allows pointing out values of interest, it is not possible to emphasize on multiple values per axis. To achieve this, the focus+context axis would have to be extended to be able to handle multiple foci. However, even then, only a limited number of different events can be highlighted via a focus+context axis. As a further possibility to represent events that are interpreted as interesting values or points in the data space, automatic brushing is suggested. In VisAxes, automatic brushing has been implemented as an action that changes the color of the graphical primitives representing values of interest (see Figure 5.15). To ensure that events are actually highlighted, it is important that the brush color is used exclusively for brushing purposes. Even though changing the color of graphical primitives is an adequate way of highlighting events in visualizations that use data representations based on line segments or dots (e.g., TimeWheel or Scatter Plot Matrices), for plot-oriented techniques

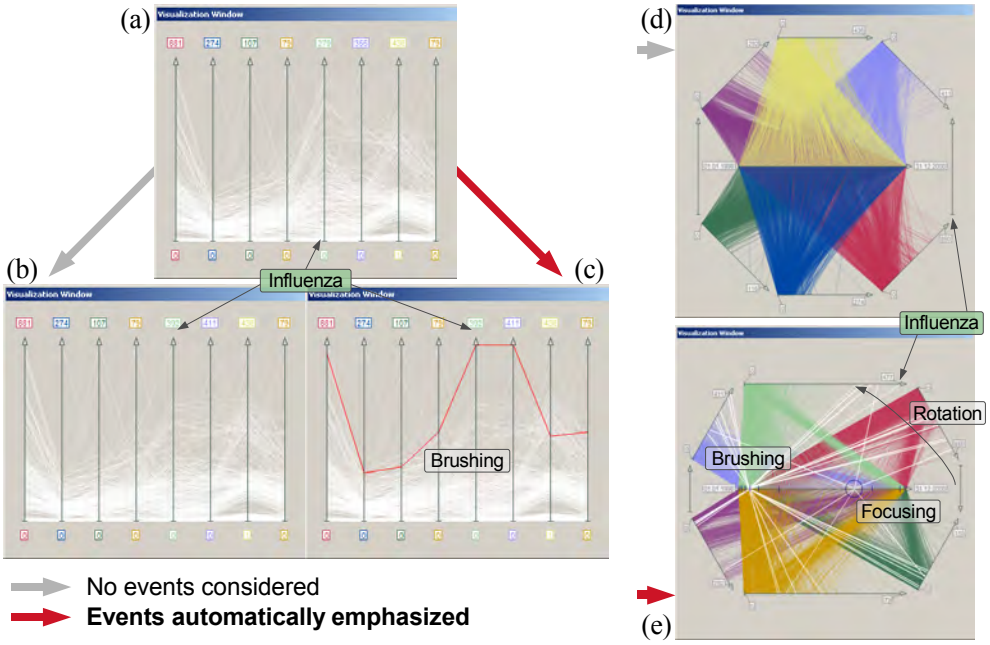


Figure 5.15.: Examples of event representations in VisAxes.

like the MultiComb, the brushing should exploit a further visual variable to achieve the emphasis. In addition to changing color, the stroke width of plot segments can be increased to highlight interesting data values. This should not be done in visualizations that use lines or dots, since occlusion of information is likely to occur.

All previously explained visual cues are useful options for the representation of event instances. It is also common to all suggested event representations that only a limited number of different event types can be communicated to the user. For a better understanding of how automatic brushing and the focus+context axis can be used to emphasize on events, Figure 5.15 shows some pictures of an eVis visualization session with automatically adjusted visualizations: A user is assumed whose interest regards days with a higher number of cases of influenza. The user applies Parallel Coordinates to analyze a dynamic health dataset (see Figure 5.15 (a)). If new data are added, the visualization is updated to reflect the changes. However, possibly interesting data entries are not emphasized (see Figure 5.15 (b)). On the other hand, considering events allows highlighting relevant data via automatic brushing (red line) (see Figure 5.15 (c)). The interesting parts of the data can be intuitively perceived.

Interesting attributes If event instances are interpreted as attributes of interest (see Figure 4.11 (a)), which is the case for attribute events, they are represented differently in VisAxes. Basically, pointing out attributes of interest is based on the focus+context capabilities of the introduced radial axes arrangements TimeWheel and MultiComb. If

an attribute event is detected, these axes arrangements can be automatically rotated such that an attribute of interest is brought to the respective focus region (see Figure 5.15). To strengthen the emphasis on the attribute of interest, a further visualization parameter is adjusted: The amount of focus magnification is increased, so that the axes in the focus region are provided with more display space making them visually more present. If multiple attributes are considered to be of interest, the axes reordering facilities of VisAxes can be used to bring further axes of interest close to the focus regions of the axes arrangements. If events have been associated with different importance values, events with a higher importance should be moved closer to the focus region. The previously described adjustments (rotation, magnification, and reordering) have been realized as gradual change (i.e., as a process). It is important to use a process in this case, since rotating and reordering the axes arrangement instantaneously would make it difficult for users to maintain their mental map. Figure 5.13 (a) shows a series of screen shots of how an attribute is brought to a dedicated focus position.

To demonstrate the use of automatic rotation, Figure 5.15 illustrates the benefit of event-based visualization for the case that a new dataset is opened for visualization (a 2D TimeWheel for this case). Instead of providing a standard representation (see Figure 5.15 (d)), taking events into account allows rotating interesting attributes (influenza in this example) automatically into the focus. Additionally, data items that are associated with interesting events can be brushed (white lines) and one event instance (e.g., the one occurred last) can be focused using the focus+context axis (see Figure 5.15 (e)). By these automatic parameter adjustments, relevant data are highlighted and users are guided in the data analysis.

A disadvantage of the suggested integration of attribute events is that events, even though possibly of different types, are represented in the same way. Additional visual hints are required to accomplish a communication of the event type. This could be achieved, for instance, by attaching additional labels to the axes representing attributes of interest. As an alternative to considering attribute events via rotations and scaling, it is also possible to take them into account at the level of the filtering step. An action that performs an automatic projection of only axes that display relevant attributes can be used for this purpose.

Interesting intervals Next, an event representation will be described that addresses events that are interpreted as interesting intervals (see Figure 4.11 (f)). In this case, certain subranges of an attribute's value range are of interest. Although data not falling into these subranges can be filtered in the filtering step, for incorporating such events into VisAxes, it is suggested to utilize the functionality of the developed scroll axis. A subrange of interest can be easily focused by adjusting the position and extent of a scroll axis' slider. A process has been implemented that accomplishes this task: If an interesting subrange must be emphasized, the process switches to a scroll axis representation and gradually adjusts the slider to match the interesting subrange. Alternatively, it is also possible to consider a certain neighborhood of the subrange of interest. This is particularly useful in cases where the subrange of interest is narrow with respect to the whole value range of the attribute (see Figure 5.16). In any case,

when using scroll axes to represent relevant subranges, it is not possible to consider multiple event instances per axis. To achieve this, either the scroll axis must be extended with further sliders, or a hierarchical axis can be used. For the latter case, it would be necessary to calculate an artificial two-level hierarchization of the original value range under special consideration of the subranges of interest that occurred. Under this assumption it would be possible to automatically open segments of the hierarchical axis to show subranges of interest in full detail, whereas the rest of the data is represented at a higher level of aggregation.

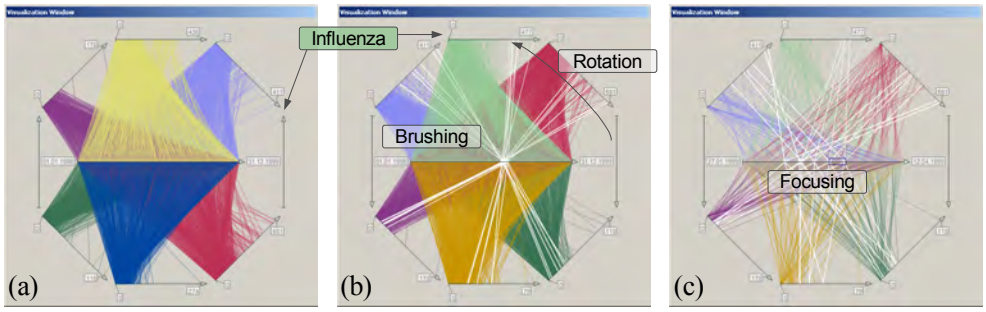


Figure 5.16.: Different levels of emphasizing on events.
(a) No consideration of events; (b) Events have been highlighted via brushing and appropriate rotation; (c) A subset of data containing event instances has been focused using a scroll axis.

Considering events to increase performance This paragraph details on the use of events for addressing performance issues. Since VisAxes aims at providing a high degree of interactivity, it must be ensured that the visualizations are drawn at interactive response rates (at least ten frames per second). It would make no sense to provide interactive rotation of axes arrangements, if every repaint of the view took several seconds. To achieve interactive response rates, VisAxes implements the rendering in two steps. In the first step, which is completed within interactive response time, only a preview is rendered. In a second step, which is running in a concurrent background thread, the full representation is calculated. To render the preview, not the whole dataset, but only a subset of the tuples is drawn. Besides using a simple sampling strategy, also knowledge about occurred events is taken into account to determine which tuples should be drawn. Apparently, tuples that are bound to event instance are important to be visible at all times; they must not be omitted when rendering the preview. In this sense, the preview visualizes tuples bound to events first and then performs a sampling of the remaining tuples. Under the assumption that the number of event detections is magnitudes smaller than the number of repaints (which is usually the case), it can be guaranteed that the most important information (i.e., the events) is always shown at interactive response rates.

Summary on event representations The implicit event representations presented previously can be summarized as:

- Pointing out via markers,
- Highlighting via brushing,
- Emphasizing via appropriate rotation and scaling,
- Focusing via focus+context axis, and
- Focusing via scroll axis.

These examples of event-enhanced axes-based visualizations give evidence of how visualization can benefit from integrating events. On the one hand, the visual representations gain from the automatic adjustment of visualization parameters; on the other hand, more technical issues like performance can be addressed using knowledge that events occurred in a dataset.

Although not specifically stated, the developed axes-based visualizations can also be applied for explicit event representation. If a collection of events is transformed to a new relational data set (as described in Section 4.6.3), any of the suggested axes-based representations can be used for its visualization. This is possible, since the introduced techniques are capable of visualizing multiple (event) attributes. The requirement of visualizing categorical data attributes (specifically the event type) has been addressed by adapting appropriate methods known in literature [Rosario et al., 2004], [Schmitz, 2005].

Approach Remarks

This section was dedicated to the description of axes-based visualizations that have been extended with event-based concepts. The presented visual representations make use of the toolkit VisAxes [Tominski et al., 2004], which provides several interactive axes (scroll axis, focus+context axis, hierarchical axis) and various axes arrangements[8] (e.g., TimeWheel, MultiComb, KiviatTube, Parallel Coordinates). All described visualizations allow a high degree of interactivity through a rich set of visualization parameters. Besides focus+context and linking concepts, further enhancements like labels, markers, and additional views have been incorporated. All in all, the proposed interactive axes-based visualizations are promising tools for supporting the exploration of multivariate data. With the integration of the proposed techniques into the framework eVis, events detected in the visualized data can also be considered. Actions and processes have been developed that automatically adjust certain visualization parameters to achieve a highlighting of events in the visual representation. This is particularly useful to improve the expressiveness, effectiveness, and appropriateness of axes-based visualizations. Indeed, this claim has to be evaluated in the future. Especially the usefulness of the different suggested event representations as well as the benefits of 3D axes arrangements compared to 2D visualizations should be assessed in future work (e.g., by considering [Brath, 1997]).

[8] More details on the introduced 3D axes arrangements can be found in [Holzhüter, 2005].

5.4.3. Data and Events Represented in Time and Space

In contrast to the previous sections, in which only implicit event representations have been described, this section concerns both explicit and implicit event representations. Yet, the focus of this section is on introducing the explicit event representation that has been implemented in the toolkit *TimeMap*; implicit event representations will be considered only briefly. Analogous to the previous sections, the descriptions given next will follow the procedure of stating general concern and basic visual representations first and explaining event-related extensions second.

Since time-related data that have been collected in a geographic frame of reference (i.e., spatio-temporal data) are relevant in a wide range of application scenarios, it is important to address these data with specialized visualizations. The techniques described in the previous sections (e.g., TableLens or Scatter Plot Matrix) can principally be used to represent spatio-temporal data. However, these techniques do not treat spatial dimensions as special attributes, but as simple numeric attributes among others. Although this is reasonable for the sake of generality, the visual representations are not always easy to interpret, especially with respect to the question of how the data evolve in time and space. In contrast to that, the TimeMap toolkit provides visualizations that specifically address the exploration and analysis of spatio-temporal data.

Choropleth maps are commonly used to represent geo-referenced data. In choropleth maps, the spatial frame of reference is represented as a map and actual data values are encoded to the color of geographical regions. To visualize temporal aspects, a series of choropleth maps, each representing a single time step, can be shown in a successive manner (e.g., as animation or slide show). This allows an easy interpretation of the spatial dependencies of the data and the evolution of the data over time. However, only a single time- and space-dependent attribute can be visualized using choropleth maps, multivariate data are not supported. Moreover, certain problems arise when using animated visualizations. Although these problems have already been discussed (also see [Tversky et al., 2002]), it makes sense to recall a major drawback of such visualizations: When using animation, it is difficult to compare data of different time steps. For doing so, users would have to switch back and forth between different images, resulting in an increase in the cognitive effort required to understand the data.

Therefore, the TimeMap toolkit adheres to static representations of multivariate spatio-temporal data. It is the goal to represent multiple attributes with respect to multiple time steps within a single map-based view. To achieve this goal, the TimeMap toolkit implements an interactive map display and novel 3D glyph-based data representations. The TimeMap toolkit has been incorporated into the framework eVis to mutually take advantage of the developed map-based and event-based concepts. In particular, the TimeMap toolkit has been extended with a special visualization technique that allows an explicit representation of event instances in time and space. In turn, the visual methods of TimeMap can be automatically parameterized according to occurred events.

Parts of the following sections have been published in [Tominski et al., 2003b], [Tominski et al., 2005c], and [Tominski et al., 2005b].

General Concept

Three basic tasks have been identified that must be accomplished by the TimeMap toolkit to enable the visualization data and events in time and space. These three tasks are:

1. Represent the spatial frame of reference as map,
2. Create visual representations of the temporal aspects of the data, and
3. Embed the representations of time (2.) into the representation of space (1.).

As already indicated, the representation of the spatial frame of reference is a prerequisite to an intuitive understanding of spatio-temporal data. In the TimeMap toolkit, the spatial frame of reference is represented as an interactive map; temporal aspects of the data are visualized via novel 3D glyphs. Embedding these glyphs into the map display allows a comprehensive visual exploration of spatial and temporal aspects of time-dependent geo-referenced data. In the following paragraphs, detailed descriptions of the developed map display, the 3D glyphs, and their combination with the map display will be given.

Interactive map display The map display of the TimeMap toolkit has been designed to represent data collected in a reference space made up of two spatial dimensions. It is assumed that the data to be visualized constitute an indirect[9] relation to spatial objects. In other words, the map is built up of geographic areas, each of which associated with a unique identifier. The actual data refer to these identifiers to describe the spatial dependencies. To represent the spatial frame of reference, two-dimensional geographic areas are represented in a 3D presentation space, in which the map can be arbitrarily zoomed and rotated using common 3D interaction (see [Henriksen et al., 2004]). Users that prefer a two-dimensional map representation can switch easily to the desired map projection.

The map display developed in the TimeMap toolkit uses a hierarchically structured description of geographic realties. In particular, the map relies on the hierarchical segmentation of states into smaller administrative units like federal states, counties, and municipalities (see Figure 5.17 (a)). This hierarchical structure allows users to choose between fine-grained visualization at the spatial level of municipalities or more and more aggregated, and hence less loaded visualizations at higher levels of the map hierarchy. In this sense, the developed map supports semantic zooming of the represented data. Apparently, adequate aggregation methods must be used to calculate data for different levels of the aggregation hierarchy (see [López et al., 2005]). Even though the simple data considered here were easy to aggregate, the aggregation of arbitrary data is generally a challenging task (e.g., because of missing values or semantic inconsistencies).

[9] An indirect spatial dependency is characterized by the fact that data and space are linked via identifiers (e.g., ZIP codes) that relate to spatial objects. Contrary, a direct spatial dependency is described by relating data directly to spatial coordinates (i.e., points in space).

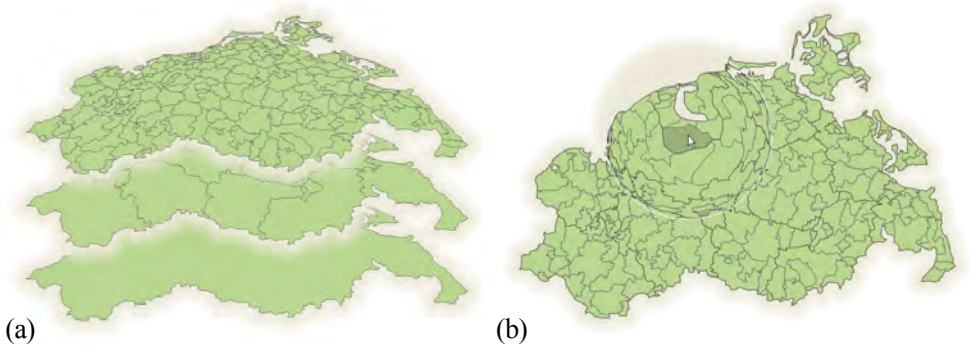


Figure 5.17.: The map display of the TimeMap toolkit.
(a) The different levels of the map hierarchy; (b) The geographic fisheye lens incorporated in the map display.

A difficulty that arises when visualizing data at finer-grained levels of the map hierarchy is that the sizes of the map areas tend to become very small, especially if the map has been zoomed out (e.g., to get an overview). This makes it hard for users to identify certain geographic areas. To alleviate this problem, a geographic fisheye lens [Rase, 1997] has been incorporated into TimeMap. Users can apply this lens in an interactive fashion to magnify a local focus region without loosing the overview of the whole map (see Figure 5.17 (b)). It is worth mentioning that applying the fisheye lens requires no additional computational effort on the CPU. Since the lens has been implemented as a vertex shader, all computations that are necessary to distort the map geometry within the scope of the lens are performed by the graphics card hardware.

The developed map display with its different possibilities of interaction serves as a basis for the visualization of data and events. In the next section, 3D glyphs are developed capable of communicating temporal dependencies. These 3D glyphs will be embedded into the developed map display later on.

3D glyphs for representing time The second task is to create visual representations of the temporal dependencies of the data. To do this efficiently, the mapping of time must be in accordance with the particular time domain inherent in the data. So far, the toolkit TimeMap supports two different types of times – linear time and cyclic time[10], which are the most common ones found in today's datasets. To represent data either with respect to linear time or cyclic time, two different 3D glyphs have been developed:

- Pencil glyphs and
- Helix glyphs.

[10]Linear time extends on a straight time axis from past to future, whereas cyclic time is based on recurring temporal primitives (e.g., the seasons of the year). More details on different types of time can be found in [Frank, 1998] or [Hajnicz, 1996].

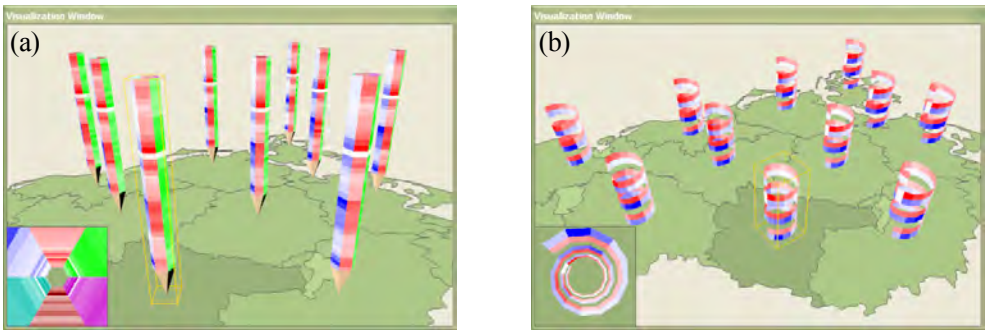


Figure 5.18.: Pencil and helix glyphs for representing spatio-temporal data.
(a) Pencil glyphs representing the number of cases of six diseases with respect to a linear time axis; (b) Helix glyphs representing two selected diseases that show potentially cyclic behavior. "Tunnel views" alleviate the visual loss of information on the back faces of the glyphs.

Since virtually everyone is familiar with the shape of a pencil, it makes sense to use it as a visual metaphor for representing linear time. This is also corroborated by the successful application of the pencil metaphor in other visualization scenarios (e.g., [Francis and Fuller, 1996]). The shape of a pencil provides multiple faces that extend from a common tip. This shape serves excellently as a 3D glyph for visualizing multiple time-dependent attributes. To map data onto a pencil glyph, each face of the pencil is associated with an attribute of the dataset to be visualized. To be able to visualize different numbers of attributes, the number of faces that are used to create a pencil glyph can vary. To encode the linear time dimension, the pencil's faces are subdivided (beginning at the common tip) according to the number of time steps in the data. Each of the resulting segments is then color-coded with respect to the particular time step and the associated attribute (see Figure 5.18 (a)).

Whereas pencil glyphs are useful for communicating data depending on linear time, helix glyphs have been developed to facilitate the representation of cyclic temporal dependencies. A spiral helix exhibits a geometric shape that allows an emphasis on the cyclic characteristics of a dataset (see [Carlis and Konstan, 1998]). To construct a helix glyph, a ribbon is created that extends in angle and height depending on the number of time steps per cycle and the number of cycles. For data that have been collected with respect to the Gregorian calendar, it makes sense to use 7, 14, or 28 segments per helix cycle. To allow users to explore their data for arbitrary cyclic patterns, the number of segments used per helix cycle can be interactively adjusted via a visualization parameter. Multiple data attributes can be visualized by subdividing the helix ribbon into narrower sub-ribbons, each of which dedicated to a separate attribute. Analogous to the pencil glyph, color-coding is used to visualize data values at the segments of the sub-ribbons (cp. Figure 5.18 (b)).

Since pencil glyph and helix glyph rely on color-coding, it is worth mentioning some aspects regarding color. Since color is easy to perceive, it is used in many visualization techniques. Certainly, color-coding has to be done properly to create expressive

and effective visualizations; incorrect color-coding can cause misinterpretations and false conclusions. A variety of articles have been published on how to use color scales appropriately (e.g., [Brewer, 1994], [Bergman et al., 1995], [Rogowitz and Treinish, 1998]). Besides choosing the right color scale, it is also important to consider an expressive mapping of data to color and to provide an expressive color legend. To increase the expressiveness of color-based visualizations (also beyond the scope of this thesis), concepts have been developed that address the latter issues. These concepts include non-linear mappings of value ranges to color scales to ease the perception non-uniformly distributed data, value range expansion to facilitate an intuitive understanding of the color-coding used, and expressive color legends based on Box-Whisker-Plots. Since a detailed explanation of these concepts is not the focus of this thesis, the interested reader is referred to [Tominski et al., 2005c].

Under the assumption that the color-coding achieves an expressive representation of data values, the implemented pencil and helix glyphs are promising visual representations to convey linear or cyclic temporal characteristics of multivariate data.

Combining 3D glyphs and map display Now that a map display can be used to represent the spatial frame of reference and that different 3D glyphs can be applied to represent data with respect to time, the last remaining task is to embed the 3D glyphs into the map display. To do so, first, either pencil glyphs or helix glyphs are created for a user-determined selection of geographic areas. Then, these glyphs can be embedded into the map display by positioning each 3D glyph at the centroid of its respective area and aligning it with the z-axis of the 3D presentation space. By doing so, the representation of the temporal dependencies is shifted from the map to a dedicated dimension in the 3D presentation space. However, the embedding in 3D implies new problems compared to a 2D representation:

- Undesired changes of the glyph view on user interaction, and
- Loss of information due to occlusion of glyphs and on back faces of the geometry.

Applying all interactions that are allowed in a 3D presentation space (e.g., zooming, rotation, translation) to both the map display and the glyphs is disadvantageous. Although such behavior is common in 3D applications, it implies perceptual inconsistencies during the visual analysis. If, for instance, users rotate the map while analyzing attributes represented on a pencil glyph, the faces of the pencils change their orientation toward the user as well, i.e., the view of data attributes changes undesirably. A solution to this problem is to unlink map interactions and glyph interactions on demand. This is achieved by aligning the glyphs not simply with respect to the z-axis, but with respect to the user's current viewpoint into the 3D scene in such a way that it can be ensured that the view of the glyphs persists while users navigate the map. To allow an interactive or automatic adjustment of the glyph view, a visualization parameter is provided that controls the rotation of the 3D glyphs. Similar to what was said about the linking of axes in the VisAxes toolkit, it also makes sense to link the rotation parameter, so that all glyphs are equally rotated. This allows users to change

the glyph view easily to analyze different attributes (for the pencil glyph) or different time steps (for the helix glyph). For detailed data analysis, it is also possible to rotate each glyph separately (e.g., to compare different attributes at different glyphs).

The second problem that must be addressed when using 3D visualizations is loss of information. Such loss occurs when 3D glyphs occlude each other and on back faces of the 3D glyphs. A first possibility to alleviate the loss of information is to improve the crude procedure of positioning glyphs at the areas' centroids. An algorithm[11] has been developed that aims at minimizing the occlusion among the 3D glyphs. This algorithm calculates the glyph positions not only with respect to the areas' centroids, but additionally takes the user's viewpoint into account. Since the viewpoint can be changed interactively, it is not uncommon that plenty of new positions have to be calculated within a second. This implies that a global resolution of all occlusion conflicts cannot be accomplished in time. Therefore, an iterative approach has been developed that locally calculates occlusion conflicts, and alters the positions of glyphs to reduce the loss of information. The algorithm is executed on each change to the viewpoint and runs until no local conflicts occur or the maximum number of iterations has been exceeded. This procedure generates improved glyph positions and yet assures interactive frame rates.

Although using a 3D presentation space allows a high degree of interaction and navigation through the data, a certain amount of possibly important information is always hidden on the back faces of the 3D glyphs. To mitigate this disadvantage, a special view – the "tunnel view" – has been developed. The "tunnel view" provides a separate 2D projection of a user-selected 3D glyph. A "tunnel view" is created by setting the rendering viewpoint directly below the selected glyph and aligning the view direction with the glyph (i.e., with the virtual time axis). Rendering a glyph from a viewpoint set up this way results in a view that reveals all data values represented by the selected 3D glyph. Figure 5.18 illustrates that "tunnel views" alleviate the back face problem, and hence facilitate the visual exploration.

The enhanced viewing and interaction methods in combination with the mentioned geographic lens and the common 3D interactions provide a rich basis for interactively exploring multivariate spatio-temporal data. How the developed techniques support the visualization of events will be described next.

Integration of events

In the previous paragraphs, different visualization and interaction possibilities of the TimeMap toolkit have been described. This section provides a discussion on how events occurrences can be considered in the developed visual representations. The main focus of this section is on explicit event representation, particularly on a technique that represents event instances in the 3D presentation space provided by the TimeMap toolkit. The introduced glyph-based visualizations can also be subject to implicit event representations via automatic parameter adjustments, which are presented in the second part of this section.

[11] Details on this algorithm can be found in [Freese, 2005].

Explicit event representation via space-time paths As indicated in Section 4.6.3, events detected in spatio-temporal data can be interpreted as interesting points in the data's frame of reference. To visualize events that are interpreted this way, the points representing event instances must be highlighted in the visualization. The interactive map display provided in the TimeMap toolkit can be used as a basis to achieve this goal. The question that has to be answered is how to explicitly mark the interesting points in the 3D presentation space. One possibility is to use small geometric primitives that are positioned at those points in the presentation space for which events occurred. Since the map display developed for the TimeMap toolkit constitutes a one-to-one correspondence between the frame of reference of the data and the presentation space (i.e., two dimensions represent space and one dimension represents time), this task can be easily accomplished. Marking all event occurrences in the presentation space results in a visualization that is useful to illustrate where in time and space events occurred. In addition to that, the most important event characteristic – the event type – can be communicated by using differently colored or differently shaped geometric primitives (e.g., spheres, cylinders, tetrahedra). This facilitates the detection of local accumulations or clusters of similar event instances. If further event parameters have to be visualized, it makes sense to consider deformable shapes as described in [Theisel and Kreuseler, 1998] and [Kreuseler and Schumann, 1999].

Yet, it is difficult to understand how event instances evolve in time and space. Therefore, the idea of space-time paths [Carlstein et al., 1978] has been incorporated. Space-time paths are nothing else than the explicit visual representation of traces of objects through time and space. Applied to the task of visualizing event instances, space-time paths are used to represent trajectories of event instances of the same type. Apparently, this makes sense only if instances of the considered event types can occur several times in the data (see Table 4.4 on page 76). Otherwise, representable traces would not exist. To generate a visual representation of an event trajectory of a particular event, successive event occurrences are linked via straight line segments. This explicit representation makes it easy for users to understand the spatial and temporal behavior of events.

For demonstration purposes, the TimeMap toolkit has been extended with the previously described concepts. The potential usefulness of the resulting visual representations will be illustrated by a simple example. For the sake of simplicity, this example is restricted to the consideration of a single event type, and hence to a single space-time path. The considered event type detects per time step where in space (i.e., for which region) the maximum number of cases of influenza occurred

$$maxInfluenza = \langle x \mid \forall_T y : (x.Date = y.Date) \rightarrow (x.Flu \geq y.Flu) \rangle_T.$$

In the visualization depicted in Figure 5.19, instances of this event type are shown as small red spheres. Successive event occurrences are explicitly connected using a space-time path, which is represented via white line segments. However, due to the common problems of perception in a 3D presentation space, it is difficult for users to discern to which regions and which time steps the event instances belong. Therefore, in addition to the red spheres, instances of the logical negation of the considered event type (i.e.,

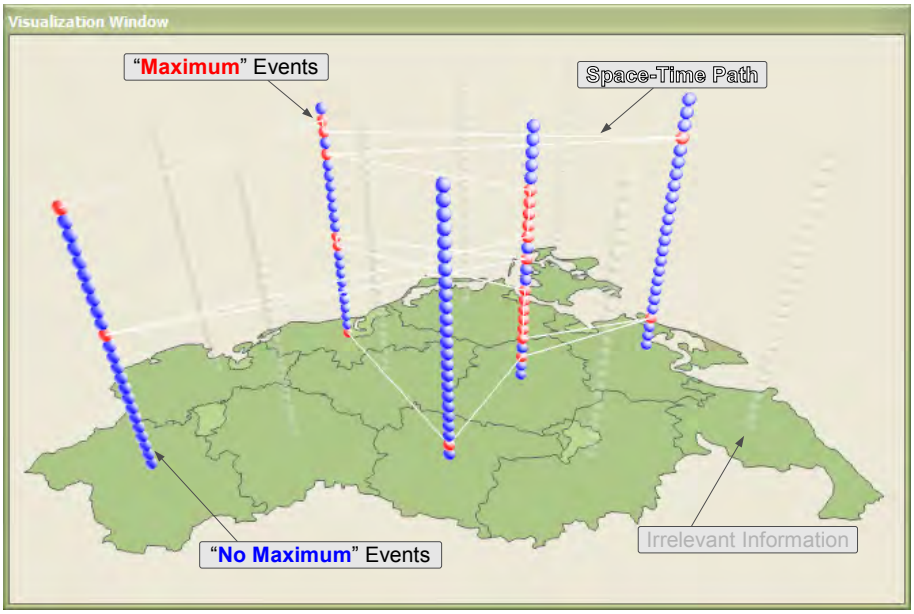


Figure 5.19.: Explicit representation of maximum events using space-time paths. The figure shows how events can be represented explicitly in the TimeMap toolkit. Red spheres visualize maximum events, whereas blue spheres depict that no maximum occurred. A space-time path is used to make the evolution of the maximum events through time and space visually explicit. Irrelevant information is faded out.

a maximum did not occur) are represented as small blue spheres. As a result, the time axes extending perpendicularly from each region of the map are completely filled up with either red or blue spheres. Although this eases the understanding of the presented space-time path significantly, display space may be wasted by the representation of the "not" events. This can be dealt with by fading out the blue spheres of regions that do not show any red sphere, meaning that a maximum has never occurred for that region. The result of conducting the steps described previously is a visualization that explicitly represents event instances, visualizes the trajectory of the event instances through time and space, and is cleaned of irrelevant information (see Figure 5.19). All concepts available in the TimeMap toolkit (e.g., 3D interaction or enhanced glyph positioning) can be applied when visualizing event instances explicitly. With respect to the considered maximum event, it is now arguably possible to find a potential way of how influenza has spread in time and space.

It must be mentioned that the notion "successive event occurrences" used in the previous paragraph bears some sign of restriction. In particular, this means that event instances have to occur at distinct points in time to be representable as a space-time path. Events that do not obey this restriction (i.e., event types for which multiple event instances may occur per time step, for instance, at different locations in space), cannot

be represented using a basic space-time path. This becomes clear when imagining the construction of a space-time path. Assumed a partial path has already been created. At a certain point in time, i.e., at the time step for which multiple event instances of the considered event type occurred simultaneously, it is not fully determined to which of the event instances the path should continue. To cope with this problem, space-time paths have to be generalized such that they allow a split in the path. From the theoretical point of view, generalized space-time paths can be modeled as acyclic directed graphs. Their representation can be easily implemented by simply adding multiple path segments at time points that require a split. The question that remains is when to join a split path. If a time step exhibiting multiple event instances (which requires a split in the path) is succeeded by a time step at which only a single event instance occurred, the apparent join is easy to perform, and also comprehensible to the users. However, if a time step that required a split is again succeeded by multiple event instances, it is not clear how to proceed with the creation of the space-time path. Finding a solution to this problem could be an interesting task for future work.

Implicit event representation The 3D glyphs and the 3D map display available in the TimeMap toolkit are controlled through various visualization parameters. Besides of user interaction, these parameters can also be adjusted automatically as response to the occurrence of events. Examples of implicit event representations that are useful to apply in the TimeMap toolkit are briefly described next.

Glyph adaptation The 3D glyphs used in the TimeMap toolkit can be subject to several automatic adjustments. A first possibility to implicitly represent the occurrence of an event is to increase the height of a glyph. By changing the size of glyphs, a visual channel is addressed that is perceived preattentively, meaning that bigger glyphs immediately attract the attention of the users. To give a second example, the rotation of pencil glyphs can also be useful to adjust. If, for instance, an attribute event occurred, it makes sense to rotate all pencil glyphs such that the attribute for which the event occurred is facing toward the user, and not represented on the back face of the glyphs.

Since the visualizations of the TimeMap toolkit are based on color-coding, it makes sense to adjust the color scale according to occurred events. The advantages of special color scales for highlighting purposes have already been described in [Bergman et al., 1995] and [Tominski et al., 2005c]. Applying such color scales can also be beneficial to implicit event representation.

A more data-centric possibility to improve the data analysis via events regards the automatic adjustment of the introduced helix glyphs. Specifically, the number of data values encoded per helix can be subject to automatic adjustment. As already indicated, this parameter is crucial for finding cyclic patterns in the data. If event types that detect cyclic patterns could be defined, it would be possible to adjust the number of data values encoded per helix cycle automatically. Certainly, the required event type has to be defined by a visualization expert.

Lens positioning To emphasize regions of the map for which events occurred, the geographic fisheye lens available in the TimeMap toolkit can be used. The lens parameters position and scope can be automatically adjusted such that the lens is positioned with respect to the region of interest and the lens scope covers the considered region's bounding box.

Tunnel view for highlighting A last option to represent events implicitly is to make use of the "tunnel view". The glyph of a region for which an event occurred can be automatically selected for the tunnel view. Furthermore, the "tunnel view" can be automatically zoomed such that interesting data values are visible at the outer border of the "tunnel view".

This brief description of examples of how event instances could be implicitly represented in TimeMap shows that there is great potential to gain benefit from event-based concepts. However, as of the writing of this thesis the described implicit event representations have not yet been implemented as actions or processes. Therefore, the previous examples are considered suggestions only. Nonetheless, they should be easy to implement, since all necessary visualization parameters are accessible via the visualization interface of the framework eVis.

Approach Remarks

In this section, the TimeMap toolkit has been introduced. The major concern of the described techniques is the visualization of data and events in time and space. Two novel embeddings of 3D glyphs into an interactive map display have been described. The glyphs address different temporal characteristics of the data: The pencil glyph visualizes data with respect to a linear time axis, whereas the helix glyph considers cyclic characteristics of the data.

In addition to the visualization of spatio-temporal data, the TimeMap toolkit allows the explicit representation of events detected from such data. The concept of space-time paths has been incorporated into the TimeMap toolkit to allow an intuitive understanding of how events behave in time and space.

Although data and event have so far been represented separately, it makes sense to investigate a combination of data visualization and explicit event representation in the future. The question that needs to be solved is how to link the space-time path concept with the glyphs used for the data representation.

5.5. Chapter Summary

This chapter was dedicated to the description of the event-based visualization framework eVis. Based on an analysis of general visualization requirements and specific event-related requirements, the architecture of eVis has been developed. Basically, this architecture reflects the major parts of the generalized model of event-based visualization. In particular, the eVis framework defines interfaces for the event specification, the event detection, and the event-based visualization. These interfaces are implemented

by different components like, for instance, as a visual editor for the event specification. Three different visualization approaches that implement the event-based visualization interface of eVis have been explained. Specifically, implicit event representations in a TableLens and in axes-based visualizations have been described. In addition to that, explicit event representations for events that were detected in spatio-temporal data have been incorporated into the TimeMap toolkit. The presented visualization approaches give evidence that event-based visualization is a promising way to support the visual analysis of multivariate data.

A momentary disadvantage of the implicit event representations as described in this chapter is that particular actions and processes are tied to a predefined set of event types. In other words, there is a lack of user definable reactions to events. A solution allowing dynamic binding of event types to actions and processes provided by a visualization module is more desirable. As already explained, a prerequisite to this is that a general formal model of parameterizable information graphics must be developed. Only if the semantic implications of parameter changes are adequately expressed by such a model, is it possible to develop a formal interface that allows users to define arbitrary actions or processes. Apparently, an appropriate user interface must also be designed. This could be accomplished by extending the management component of eVis, so that associations between event types and actions/processes can be handled more flexibly. Similar to the suggested model of user-centered event specification, concepts must be developed that support users in the definition and binding of actions and processes.

6. Potential Event Enhancements for Graph Visualization

In the previous chapter, visualization approaches have been described that are available in the framework eVis. However, event-based visualization is not limited to only a TableLens, axes-based visualizations, or space-time paths. Quite the opposite, the event-based concepts as developed in this thesis can be applied in many visualization scenarios to ease the visual analysis and to gain deeper insight into the investigated data. To substantiate this claim, it will be explained in this chapter how the visualization of graphs can benefit from considering events. Graph visualization has been chosen, because nowadays it is more and more important to analyze not only data, but also the structure of data and information.

The reader should note that the concepts described next have not been incorporated into the framework eVis. In this case, it makes more sense to do it the other way around, i.e., to incorporate concepts developed for the eVis framework into graph visualization approaches.

6.1. Coordinated Graph Visualization

So far, the focus has been on the event-enhanced visualization of multivariate data of relational form. In contrast to that, the approach described in the following addresses the visual exploration of graph structures and how their analysis can be facilitated via event-based concepts. The major challenge when considering graph structures is coping with larger graphs. Taking events into account is a promising way to alleviate this challenge. The novel concepts to be introduced are based on previous research by [Abello and Korn, 2002], [Abello, 2004], and [Abello et al., 2006]. Their ideas and concepts have been taken up and extended with several useful techniques as described in [Tominski et al., 2006a]. The goal of the following discussion is to illustrate how the developed extensions can be combined with event-based concepts.

Before describing visual methods for analyzing graph structures, relevant theoretical foundations will be explained briefly. An undirected (directed) *graph* $G = (V_G, E_G)$ consists of a set of vertices V_G and a collection of unordered (ordered) pairs of vertices called edges $E_G \subseteq V_G \times V_G$. A rooted tree is an undirected connected graph without cycles in which one particular vertex is marked as the root. To enable the visual analysis of large graphs, hierarchical clustering can be applied to create a hierarchical partition of the graph's vertices [Herman et al., 2000]. This partition, also called clustered (or hierarchical) graph, is commonly represented by a rooted *hierarchy tree* $T = (V_T, E_T)$ whose leaves are in one-to-one correspondence with the original graph's

vertices $\texttt{leaves}(T) = V_G$. Each non-leaf vertex in the hierarchy tree represents the input subgraph induced by the set of its descendant leaves, meaning that the non-leaf vertices of the hierarchy tree represent the original graph at different levels of aggregation[1]. The reader should note that the edges E_T of the hierarchy tree do not represent aggregated edges of the underlying graph, but reflect the outcome of the applied hierarchical clustering.

Visualizing a graph, particularly a larger one, is a challenging task. Since a graph is an abstract structure, it is necessary to calculate a spatial layout of the graph's vertices first. A variety of methods can be applied to compute a graph layout that obeys to certain aesthetics criteria and can be rendered on a computer display [Battista et al., 1999]. The most common way of visualizing graph layouts is to represent vertices as geometric primitives (e.g., dots or spheres) and to link two vertices via a line or an arc if they share a common edge. However, when facing larger graphs, a simple representation of vertices and edges is usually insufficient. Only if users are supported in exploring and navigating the graph at hand, visual methods can show their full potential. One way to achieve the required degree of support is to make use of the hierarchy tree T.

Therefore, the graph visualization tool CGV (Coordinated Graph Visualization) has been developed. CGV provides diverse interactive visual methods for the exploratory analysis of graph structures. Since CGV has been designed in accordance with the suggestions made in [Abello et al., 2006], it follows a multiple views approach to visualize graphs. The main concern of CGV is to drive the navigation of larger graphs by providing views of the hierarchy tree – the *tree view* and the *hierarchy view* – in conjunction with aggregated views of the underlying larger graph – the *graph view* and the *matrix view*. Figure 6.1 illustrates how the different views are combined in CGV. All provided views are coordinated in the sense that user interactions conducted in one view are also reflected in all other views. Without going into too much detail, the coordinated views available in CGV will be briefly described in the following:

Tree view The tree view (see Figure 6.1 (c)) represents the vertices V_T and the edges E_T of the hierarchy tree T. The main purpose of the tree view is to provide textual information associated with the vertices V_T. Since this view relies on the concept of tree views so prevalent in common user interfaces, they are easy to understand and to operate. The tree view of CGV allows the interactive expansion and collapsing of vertices of the hierarchy tree to change the level of aggregation of the representation of the underlying graph G. In addition to that, CGV's tree view incorporates the overview+detail and the focus+context concept. It is possible to zoom out the tree view to get an overview or to apply a fisheye transformation to blend in details without loosing the overview; this approach is referred to as Fisheye Treeview [Tominski et al., 2006a]. In conclusion, the tree view is used to represent textual information (e.g., vertex labels) and to drive the navigation of the underlying graph G.

[1] The notion of level of aggregation is formally defined as *anti-chain*. For more details on anti-chains in the context of graph visualization, the interested reader is referred to [Abello and Korn, 2002] or [Abello, 2004].

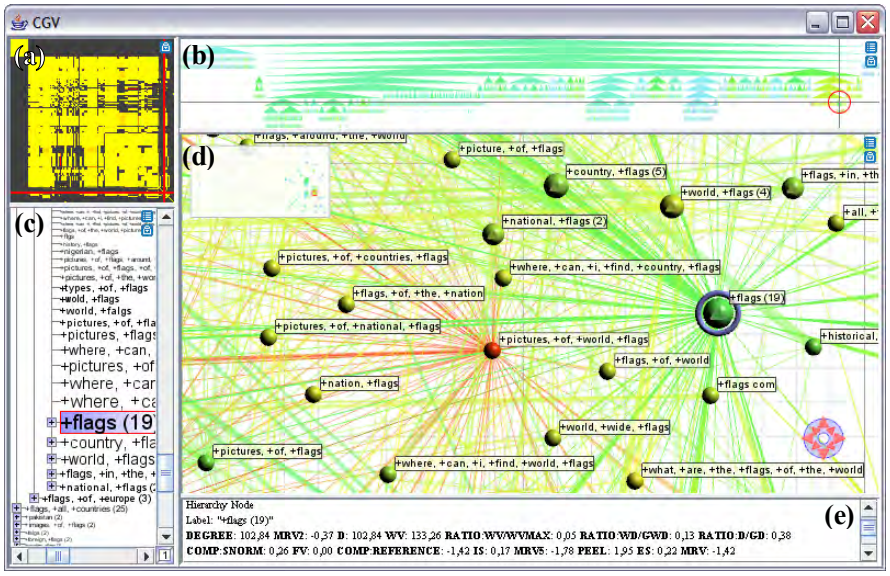


Figure 6.1.: The coordinated graph visualization tool CGV.
The user interface of CGV has been designed in accordance with AskGraphView [Abello et al., 2006], which is being developed at ask.com.

Hierarchy view The hierarchy view (see Figure 6.1 (b)) provides a complementary view of the hierarchy tree T. In contrast to the previously described tree view, the goal of the hierarchy view is to represent the entire hierarchy tree at a glance. This is achieved by switching to a vertical layout of the vertices of T. This layout can be visualized using either triangles or dots & lines. Although textual information can no longer be represented in the hierarchy view, it is possible to color-code derived graph-theoretic statistics (e.g., vertex density) or any attribute associated with the vertices of T. The hierarchy view also incorporates overview+detail and focus+context concepts [Tominski et al., 2006a]. In conclusion, the hierarchy view of CGV is responsible for visualizing the hierarchy tree and associated or derived vertex attributes at a glance.

Graph view In contrast to the previous views, which focus on the representation of V_T in conjunction with E_T, the graph view (see Figure 6.1 (d)) is dedicated to the visualization of the underlying graph G at a certain level of aggregation. Although the graph view depicts a subset of V_T, which are aggregated nodes of the underlying graph G, it does not represent the edges E_T (as the tree and the hierarchy view do), but edges that are aggregations of the original edges E_G.
In CGV it is assumed that an adequate spatial layout of V_T has been computed. This spatial layout is visualized by representing vertices as small spheres. Besides providing additional labels with the spheres, it is also possible to encode arbitrary

vertex data to the color of the spheres. To represent edges, cone-shaped links are established for vertices that share a common edge. The special cone shape is a useful visual cue for representing "how strong" (i.e., the edge weight) two vertices are related via an edge compared to all other edges connected to either vertex. To support the navigation of the visualized graph, several interactive means, including zooming and panning the layout, expanding and collapsing vertices (to adjust the level of aggregation), or traversing the represented graph in an edge-oriented manner (i.e., to travel successively from node to node) have been incorporated into the graph view. Additionally, smooth and efficient viewport animation as introduced by [van Wijk and Nuij, 2004] has been implemented. The orientation in the spatial layout is facilitated by providing a separate overview of the whole layout on demand. To cope with the visual clutter that can occur when larger graphs are visualized, the representation of the edges can be faded out in a focus+context manner. Clearly speaking, if the user focuses on a vertex (e.g., by using the mouse cursor), only the k-neighborhood of the focused vertex is represented, where k usually equals 1 or 2. In conclusion, the graph view is the main view of CGV, meaning that most graph exploration and analysis tasks will be conducted in this view.

Matrix view The last graphical view provided in CGV is the matrix view (see Figure 6.1 (a)). This view can be seen as a view complementary to the graph view. The matrix view represents the interrelation in a subset of V_T as an adjacency matrix. The color of each cell of the matrix view encodes the density of the subgraph induced by the vertices adjacent to a cell, i.e., a measure that reflects the "degree of relation" between vertices. The matrix view can also be used to expand and collapse vertices of the hierarchy tree.

Text view Finally, CGV provides a view that is responsible for presenting certain vertex related information in a textual way (see Figure 6.1 (e)). This text view is also used to provide a hyperlink to further information.

To avoid redundant user interaction, all views provided in CGV are coordinated. In particular, all views can be used to trigger basic exploration-related tasks like identifying a vertex, focusing on a vertex, or expanding and collapsing a vertex. The former operations are reflected by an update of all views. In contrast to that, expand and collapse operations are only represented in views that represent a subset of V_T at a certain level of aggregation (the tree view, the graph view, and the matrix view). In these views, expand or collapse operations require replacing a vertex with its child vertices or vice versa. This ensures that all views represent a common level of aggregation. Since the hierarchy view maintains an overview of all vertices in V_T at all times, there is no need to update this view after expand or collapse operations.

It is worth mentioning, that the coordination in CGV has been implemented in strict accordance with the Model-View-Controller pattern [Gamma et al., 1994]. The hierarchy tree serves as the model, on which basic operations are defined. The different visual representations serve as views and controllers in that they represent the model and implement the defined basic operations as interactions.

Although the coordinated graph visualization described previously is a helpful tool for visually exploring larger clustered graphs, there is still room for further improvements. In particular, CGV can benefit from integrating event-based concepts. Therefore, potentially useful event-related extensions to CGV are described in the following section.

6.2. Supporting Graph Visualization via Events

CGV as described in the previous section does not contain event-related concepts. On the other hand, the event-based model developed in Chapter 4 has been designed as a general model, so that it can be applied in arbitrary visualization scenarios. To substantiate this claim, two event-related extensions of CGV will be described. The first extension regards an advanced filtering mechanism that is useful not only to declutter dense graph visualizations, but also as a visual tool to specify event types with respect to vertices of graphs. A second extension concerns the development of novel lenses. These lenses can be used interactively during the exploration of graphs. In addition to that, the proposed lens techniques can also be used to focus the users' attention automatically on interesting parts of the graph layout. In this sense, the lenses are useful to represent detected event instances implicitly.

Visual aids for filtering and event specification

Data that are based on graph structures generally comprise vertices and edges, and also additional attributes associated with them. When analyzing such graphs with respect to certain characteristics, it can be very helpful to have the possibility to filter vertices that do or do not exhibit certain values in their attributes. This will de-clutter the visual representations, since less-relevant, i.e., filtered, vertices can be omitted; users can fully concentrate on the vertices of interest.

To allow an efficient use of filters, users must be provided with tools for defining the characteristics they are interested in. This is analogous to specifying interests as event types. The event specification formalism described in Section 4.4 is based on relational data and predicate logic formulas. Under the assumption that vertices and associated attributes are given as relational data, the event formulas introduced in that section can be adapted to describe filters for graph visualization. Accordingly, CGV has been extended with a filter model that uses specialized event formulas that allow the expression of conditions via the predicates $\leq$, $=$, and $\geq$ and their combination via the logical operators $\wedge$ and $\vee$. In CGV, basic filters describe conditions that must be met for a vertex to pass the filter. More complex filters can be created by logically combining basic filters: A logical $\wedge$ (AND) combination generates a filter that can only be passed by a vertex if it obeys all conditions; for logical $\vee$ (OR) combinations it is sufficient that a vertex passes any of the conditions involved. Since this way of describing filters can be considered a special case of the event specification formalism, it is possible to interpret the introduced filters as event types that express certain interest with respect to vertices of a graph.

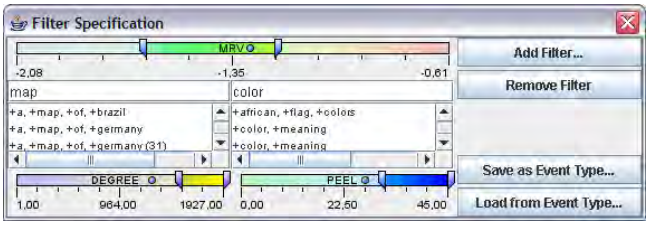


Figure 6.2.: User interface for filtering and specifying event types.

To enable users to filter vertices dynamically during the visual exploration, a visual interface for specifying filters has been integrated into the user interface of CGV. The filter specification interface (see Figure 6.2) follows the metaphor of a sieve. Accordingly, vertices have to pass the filters from top to bottom. This metaphor is modeled as a single logical *AND* combination of an arbitrary number of logical *OR* combinations that comprise arbitrary basic filters. The basic filters involved in an *OR* combination are arranged horizontally, making it easily understandable for the user that it is sufficient that a vertex passes any of the basic filters. In contrast to that, the *AND* filter is represented by vertically arranging the participating *OR* combinations. This allows users to intuitively comprehend that a vertex has to pass all *OR* combinations. To actually specify conditions of interest, the described model (one *AND* combination of several *OR* combinations) is filled with basic filters. To these basic filters belong a range slider, which can be used to define values or value ranges of interest, and a textual filter, which addresses interests expressed with respect to vertex labels. Depending on the particular needs of the application scenario, further basic filters can be easily incorporated. Figure 6.2 shows an example of a filter that can only be passed by vertices that obey the following constraints:

$$\langle v \mid v.mrv \in [-1,64 \,..\, -1,2] \wedge$$
$$(v.label \textbf{ contains } \text{"map"} \vee v.label \textbf{ contains } \text{"color"}) \wedge$$
$$(v.degree \in [1542 \,..\, 1927] \vee v.peel \in [31,4 \,..\, 45])\rangle_V .$$

Using the previously described filter specification interface, it is possible for users to dynamically filter vertices of interest. In addition to that, the specified filters can be considered as event types of interest: If users have specified their interests, it makes sense to store this information as an event type, so that in later visualization sessions the event type can be used to detect whether the same interests are contained in the data under investigation. If this is the case (i.e., event instances have been found), the visual representations can be automatically adapted to emphasize on the users interest. In this sense, the filter mechanism incorporated in CGV serves as a visual interface for specifying event types with respect to vertices of a graph. However, here the same restriction holds as mentioned for brushing techniques discussed in Section 4.4.2: It is only possible to specify interests that are already contained in the data; to specify arbitrary event types, the visual editor environment described in Section 5.3 can be used.

Implicit event representation via lenses

In the preceding discussion it has been described how event types can be specified in CGV. To gain benefit from the stored event types, it is also necessary to incorporate concepts for representing detected event instances. In Section 4.6.2 it has been stated that lens techniques are useful visual tools to represent event instances implicitly. In accordance with that, CGV has been enhanced with special lens techniques. These lens techniques can be used not only to focus the users' attention on interesting parts of the data (e.g., vertices with high degree of centrality), but also to facilitate the display of local graph information that might otherwise be difficult to grasp [Tominski et al., 2006a].

In the following, the *Local Edge Lens* and the *Bring Neighbors Lens* are presented. These two lenses are combined with a fisheye lens to a *Composite Lens* that integrates the advantages of all three lenses.

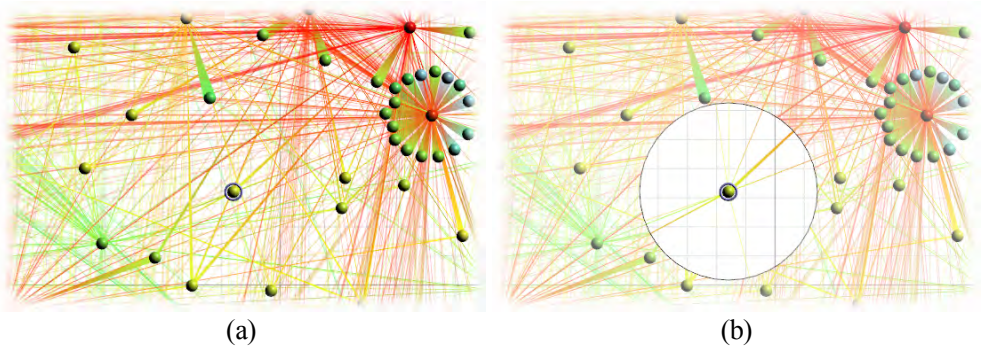


Figure 6.3.: The Local Edge Lens.
(a) Visualizing larger graphs can result in heavy clutter of the visual display; (b) The Local Edge Lens can be used to de-clutter a local display area for a vertex of interest.

Local Edge Lens The first lens to be presented addresses the heavy clutter of edges that might occur when visualizing larger graphs. As shown in Figure 6.3 (a), when visualizing non-planar graphs consisting of larger numbers of edges, the display can end up completely covered with lines. This renders the identification of edges connected to a particular vertex nearly impossible to accomplish.
Since it is essential that users easily perceive all relevant information for vertices that are associated with event instances, the Local Edge Lens[2] aims at enabling users to identify edges connected to such vertices. To achieve this goal, an alternative rendering of the graph layout is applied for a spatially confined lens region: Outside the lens, all edges are shown as usual, whereas inside the lens (i.e., the focus), only edges that are connected to vertices within the lens scope are drawn. As can be seen in Figure 6.3 (b), the cluttering of edges is removed for a local

[2]This lens is very similar to the techniques suggested in [Stone et al., 1994].

focus region that is defined by the position of a vertex of interest. Additionally, a semitransparent pane is rendered outside the lens. By this frosted-glass effect, the attention of the user is attracted to the relevant information presented inside the lens, where local edges can be discerned more easily. The Local Edge Lens is a purely graphical lens, and hence can be computed quickly by today's graphics cards.

Bring Neighbors Lens The Bring Neighbors Lens is a more complicated lens that operates on the graph layout. This lens addresses another issue that arises when analyzing larger graphs – users commonly need to compare the neighbors of a particular vertex of interest. However, neighboring vertices are not necessarily close in the graphical representation or might even be off screen. Literally, the aim of the Bring Neighbors Lens is to bring in the neighbors of a vertex of interest that is approached by the lens, regardless of their actual position in the graph layout. In other words, when the lens is moved toward a vertex (e.g., automatically by a process that reacts to the detection of an event, or interactively by the user), the neighboring vertices should be brought to the lens in a continuous fashion. This is achieved by relating the distance between the vertex of interest and its neighbors to the distance between the focal point of the lens and the vertex of interest. Specifically, if the distance between the focal point and the vertex of interest is zero (i.e., the lens is directly on top of the vertex of interest), then all neighboring vertices are brought to the lens area such that the farthest neighbor (with respect to the original layout) is positioned on the lens perimeter and all other neighbors are positioned closer to the lens center. If the vertex of interest is on the perimeter of the lens, no change is applied to the positions of the neighboring vertices. For intermediate positions of the vertex of interest (i.e., positions between the focal point and the lens perimeter), the positions of the neighboring vertices are linearly interpolated. By applying this interpolation while moving the lens toward a vertex of interest, users get an impression of the positions of neighboring vertices in the original layout. When all neighbors have been brought to the lens (the lens is directly on top of the vertex of interest), users can compare the colors and sizes of the neighboring vertices and the vertex of interest at a glance. In this sense, the Bring Neighbors Lens highlights not only a vertex of interest, but also a certain neighborhood, which otherwise might even be invisible.

However, for the calculation of the Bring Neighbors lens, it is assumed that only one vertex is within the scope of the lens. To overcome this limitation, a generalized Bring Neighbors Lens that is able to handle multiple in-lens vertices has been developed. The main idea is to calculate the attraction of neighboring vertices depending on the number of vertices within the lens scope and the respective distances to the focal point of the lens. This is accomplished by calculating, for each vertex of interest, a weight that determines how much the respective neighbor is attracted. For adequately attracted neighbors, the weights must obey the following five constraints:

1. The sum of all weights must be one.
2. If the distance between the focal point of the lens and a vertex of interest is zero, then the weight of that vertex shall be one, resulting in all neighbors being brought within the lens perimeter.
3. If a vertex of interest lies on the perimeter of the lens, then its weight becomes zero, resulting in no attraction of neighbors.
4. If two vertices of interest have the same distance to the focal point of the lens, their weights are the same.
5. If one vertex of interest has a smaller distance to the focal point of the lens compared to a second vertex of interest, then the weight of the first node is greater than the weight of the second node.

Having calculated the weights within the stated constraints, it is possible to compute the overall neighbor attraction as a weighted sum of the original one-vertex attraction case. This generalized Bring Neighbors Lens allows continuous movements of the lens without visual inconsistencies or pop-up artifacts. Figure 6.4 demonstrates the use of the Bring Neighbors Lens. It must be mentioned that still images of the Bring Neighbors Lens cannot fully convey the lens effect, because much information is communicated only when the lens is in motion (e.g., from where do vertices move in, how fast do they move compared to other vertices).

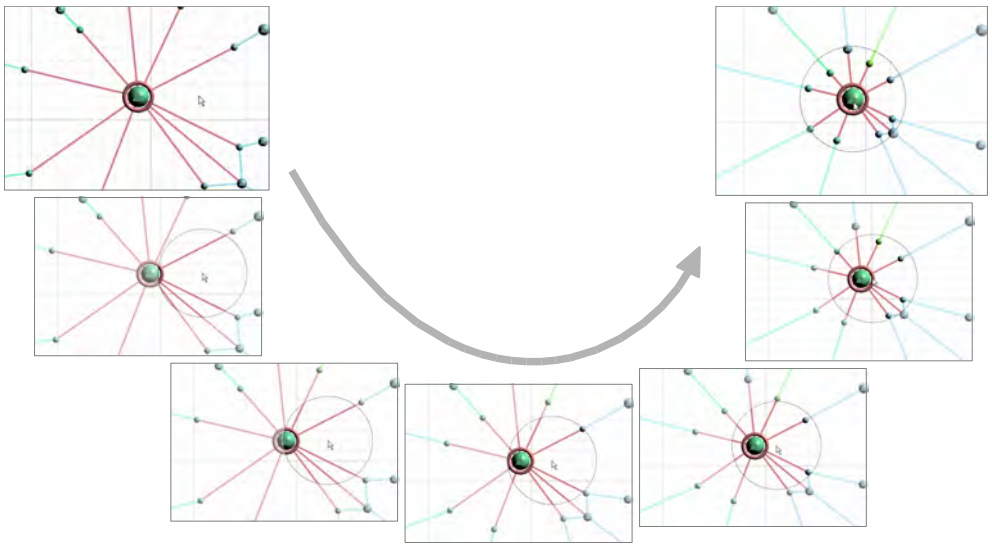


Figure 6.4.: The Bring Neighbors Lens.
The figure demonstrates the effect when approaching a vertex of interest with a Bring Neighbors Lens: Neighboring vertices are brought to the lens in a continuous fashion, so that finally the local neighborhood can be perceived at a glance.

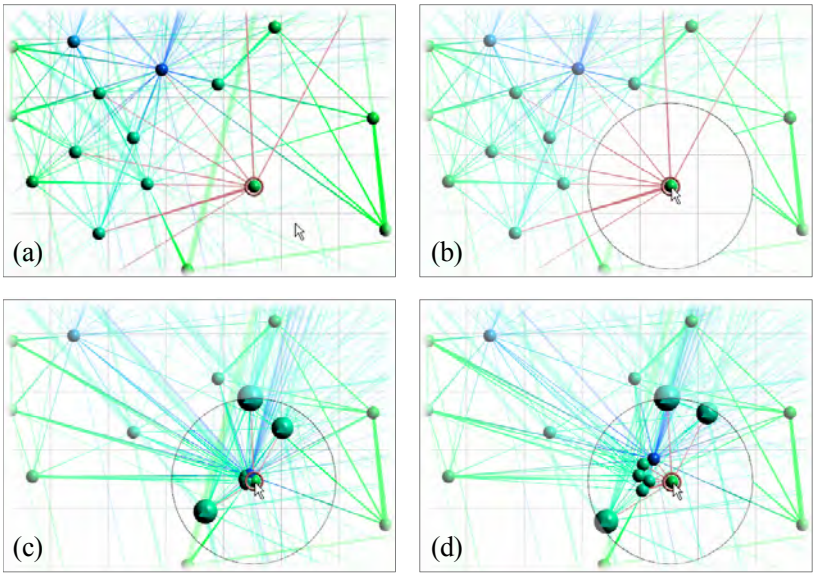


Figure 6.5.: Comparison of different lenses.
(a) No lens; (b) Local Edge Lens; (c) Bring Neighbors Lens; (d) Composite Lens, which incorporates (b), (c), and a fisheye lens.

Composite Lens The previously presented lenses work either on the representation level (Local Edge Lens) or the layout level (Bring Neighbors Lens). Indeed, it is questionable if this separation is always the best solution. Moreover, the attraction mechanism of the Bring Neighbors Lens bears a potential problem. Particularly, if a neighboring vertex is far away from the focal point of the lens while all other neighbors are close by, the closer neighbors are attracted toward the vertex of interest so much that it is impossible to discriminate the neighboring vertices from each other (see Figure 6.5 (c)). To alleviate this problem, a fisheye lens can be applied. Since a fisheye lens moves vertices away from the focal point, this spreads the neighbors attracted "too much" to the vertex of interest. To combine the previously introduced Local Edge Lens and Bring Neighbors Lens with a fisheye lens into an enhanced Composite Lens, it is necessary to determine the lens composition order. It makes sense to adhere to the visualization pipeline ([dos Santos and Brodlie, 2004], see Figure 2.1 on page 7), and consequently, apply lenses that work on the layout level (mapping step) before lenses that operate on the representation level (rendering step). Since neighboring vertices are to be attracted first by the Bring Neighbors Lens, and then have to be slightly spread by a fisheye lens, the overall order for the Composite Lens is determined: The Bring Neighbors Lens is applied first, followed by the fisheye lens second, and the Local Edge Lens last. The result of this consecutive application of lenses is called a Composite Lens. In Figure 6.5, the visual appearance of the Composite

Lens is compared to the Local Edge Lens and the Bring Neighbors Lens. Specifically, Figure 6.5 (d) depicts the effect of the Composite Lens. When comparing figures (c) and (d) it becomes clear that the additional fish eye lens separates the neighboring vertices that were attracted toward the vertex of interest. Figure (d) shows that by integrating the Local Edge Lens, the lens area is clear of edges that are currently less relevant.

The previously described lenses are useful visual tools to emphasize vertices for which event instances occurred. Since the position and the scope of the lenses can be adjusted via visualization parameters, it is easy to develop actions or processes that implement appropriate parameter adaptations. To emphasize interesting vertices via the Local Edge Lens, it is sufficient to implement an action that immediately places the lens on a vertex (e.g., a vertex with a high degree). Event types can be communicated using different colors for the lens border or differently shaped lenses. However, the latter option requires some extensions to the lens mechanism implemented in CGV. Extending CGV such that multiple lenses can be used to highlight multiple vertices simultaneously should also be pursued in the future. For the Local Edge Lens, this should be easy to accomplish, since it is a purely graphical lens. In contrast to that, the Bring Neighbors Lens alters the graph layout, making it more challenging to develop a multi lens extension. Therefore, the Bring Neighbors Lens' main purpose is to emphasize on a single vertex of interest and its neighboring vertices.

Chapter Summary

In this chapter, the graph visualization tool CGV has been described. Developed in accordance with [Abello et al., 2006], CGV provides multiple coordinated views that fulfill different purposes by representing a hierarchically clustered graph in different ways. The coordination of the different views and a high degree of interaction make CGV a useful visual tool for the exploratory analysis of larger graphs.

To substantiate the applicability of the generalized model of event-based visualization developed in this thesis, CGV has been extended with two event-related enhancements. In particular, a filter mechanism has been developed that can analogously be used to specify event types of interest with respect to vertices of graphs. A visual user interface for specifying filters dynamically and storing them as event types for later use has been incorporated into CGV. Since CGV provides immediate visual feedback when specifying or modifying a filter, this visual user interface can be understood as a means for "specifying event types by example".

Secondly, helpful lens techniques have been developed. These lenses can be used to implicitly represent event instances with respect to vertices of a graph. Commonly, event instances occur if interests defined with respect to a static graph are also detected in other graphs analyzed later in a visualization session. By taking specified event types into account, it is possible to use the developed lenses to automatically focus on relevant parts of graphs loaded into CGV for visual analysis. The suggested event enhancements are also of advantage when visualizing dynamically changing graphs. Because the views of CGV do not address dynamic graphs, this claim could not be corroborated.

In the future, further extensions with respect to event-based visualization should be investigated. It makes sense to consider additional graph-related event types (e.g., event types that take edges into account) and respective specification, detection, and representation methods. Since the Bring Neighbors Lens gives evidence that adaptations of the graph layout are useful, further layout adaptations should be invented to increase the relevance in graph visualizations by implicitly representing events. It should also be examined whether the multi-view architecture of CGV can be extended with a view for an explicit event representation.

7. Summary and Future Work

Visualization is a promising method to aid in the exploration and analysis of larger volumes of data. However, it is a demanding task to develop techniques that generate effective and expressive visual representations. A major challenge is coping with constantly growing volumes of data, which usually cause cluttering in visual representations. The second problem to mention is the lack of relevance in visual representations [Amar and Stasko, 2005]. This is caused by the disregard of user interests in current visualization techniques. Finally, current visualization approaches are rather inflexible. This is due to the fact that visualization techniques are usually custom-made to suit only one particular visualization problem [North et al., 2002]. The goal of this thesis was to develop concepts that help to solve the identified problems.

A suitable approach to **increase effectiveness, relevance and flexibility** of visual representations is to shift the interests of users into the focus. Considering the interests of users enables to focus on only relevant data items. This reduces cluttering in visual representations, and hence increases effectiveness. Relevance is accordingly increased, because less-relevant data can be suppressed. By taking the needs of users into account, it is also possible to adapt visualizations to the users' tasks at hand, which implies an increase in flexibility of visual representations.

The concept of choice to achieve a higher degree of user involvement is event-based visualization, that is, the combination of event-based methodology with visualization methods. The basic idea is to let users specify their interest by means of event types, to detect instances of these event types in the data, and to create visual representations that are adjusted according to the users' interests.

The interests of users have not yet been considered in previous event-related visualization approaches. Moreover, the literature review indicated that the available event-related visualization approaches are mostly specific to particular visualization problems, and hence cannot be applied generally. Therefore, the main concern of this thesis was to develop a novel **generalized model of event-based visualization** that considers the needs and interests of users.

Thesis summary

In this thesis, a comprehensive overview of event-related visualization approaches is given. This review of the state of the art considers not only methods from information visualization, but also concepts known in other visualization domains (e.g., software visualization). To facilitate the identification of similarities and dissimilarities as well as unsolved problems in the diversity of the reviewed approaches, a classification schema for event-related visualizations has been developed. Both a general state of the art

overview on event-related visualizations and a classification schema for such approaches are not available in literature.

The main contribution of this thesis is the development of a novel generalized model of event-based visualization. Based on an informal description of the main ideas of the model, a formalization of the model has been conducted. The three major aspects of event-based visualization – event specification, event detection, and event representation have been investigated in detail.

The developed concepts for *event specification* are based on event formulas that are tightly coupled with predicate logic and the relational data model. These event formulas enable users to describe their interests as event types. To bridge the gap between the event specification formalism and the informal interests of users, a user-centered model for event specification has been proposed. This model addresses the needs of users with different experience. To further simplify the specification of events, a visual editor environment has been developed.

To enable the *detection* of events in relational data, several formal mechanisms have been introduced. These mechanisms evaluate the specified event types to determine if portions of the data conform to the interests of the users. To achieve efficient event detection, the capabilities of relational databases and the Optimized Pattern Search (OPS) algorithm are used.

Computing expressive visual *representations* that take detected events into account is the final step of event-based visualization. Two basic concepts for representing events have been proposed – implicit and explicit event representation. Implicit event representations aim at incorporating events into visual representations by appropriately adjusting visualization parameters. Explicit event representations focus on visualizing events, rather than on representing data.

The developed concepts and methods have been implemented in the event-based visualization framework eVis. This framework has been designed in accordance with the proposed model. eVis comprises several novel visualization approaches, including axes-based techniques (e.g., TimeWheel or KiviatTube), or map-based techniques (e.g., pencil glyphs on maps or space-time paths). The implemented visualization methods take up the concepts introduced in this thesis to achieve the previously stated goals. Several implicit event representations that emphasize relevant information have been incorporated into eVis. By guiding the attention of the users toward interesting parts of the data, effectiveness and relevance of visual representations is increased. The framework eVis also accommodates visual methods for representing events explicitly in time and space. These methods are useful to achieve higher level insights by abstracting from the underlying data. Visualizing events, rather than data, is can be crucial when facing large datasets.

That event-based visualization is generally useful beyond the scope of the eVis framework has been elucidated in a discussion on the graph visualization tool CGV. This tool has been extended by a visual interface for event specification and by several advanced lens techniques that are useful to focus on relevant information.

In conclusion, the generalized model of event-based visualization is a promising means to improve visual representations. Since the developed model allows users to

describe their needs and interests, visual representations individually adjusted to the task at hand can be provided. The application of event-based visualization to different visualization problems, including multivariate data visualization, visualization in time and space, and graph visualization, gives evidence of the generality of the developed model.

Future work

Even though a wide range of concepts and methods have been developed in this thesis, still more work needs to be done to exploit the full potential of event-based visualization.

Speaking in terms of the event specification, it makes sense to investigate further abstract event types to ease the use of event-based visualization in different application scenarios (e.g., event types that consider object-oriented data or XML data). The composite events introduced in this thesis are rather basic. Their expressiveness can be improved by incorporating notions from regular expressions. Further visual aids could be developed to narrow the gap between the formal event specification and the informal interests of users. From a more theoretical point of view, it seems to be very interesting (and equally challenging) to develop methods that can be used to analyze user-specified event types for the purpose of automatically suggesting (or generating) further event types that might also be of interest to respective users. It is worth mentioning that designing visual interfaces that simplify the use of formal methods as well as deducing potential user interests from specified event types are challenges in their own right with high relevance beyond the scope of event-based visualization.

Regarding the detection of events, more effort should be spent on incorporating incremental detection algorithms. For doing so, a thorough analysis of the theoretic foundations of incremental methods known in the field of database research has to be conducted. This is of particular importance when facing data streams, in which events can only be detected with respect to a limited view of the data. Theoretical investigations of the proposed abstract event types should be conducted to identify further possibilities to improve the event detection efficiency. It is strongly recommended to work closely together with experts in databases and data mining to find proper answers to the raised questions.

From the visualization point of view, there are several aspects that should be examined in the future. First of all, the flexibility of the developed implicit event representations should be improved. Similar to the event specification, there should also be an interface that allows users to specify what interests should be emphasized in what manner. Whereas a simple mechanism for dynamically binding event types to actions/processes should be easy to implement, developing an entire specification language for actions and processes at the granularity of visualization parameters is by far a more challenging task. A prerequisite to achieving such flexibility is the development of a formal model of interactive visualization that includes aspects like graphical encodings and interaction techniques as well as visualization parameters, their semantic role in the visual representation, and possible dependencies among them. In other words, a model is required that describes the visual effects of parameter changes and also

which dependencies exist between visualization parameters. Even though this problem is being addressed by many researchers, no such model exists in literature. If a proper model is found in the future, it should be possible to develop tools that allow users to create their own implicit event representations.

Aspects of explicit event representations should also be addressed in future work. Specifically, it should be investigated which attributes of event instances are useful to visualize in order to facilitate higher level insight into the underlying data. Depending on the outcome of this analysis, further explicit event representations could be developed.

Perceptual issues of event-based visualization have been discussed only briefly in this thesis. In future work, the implications of the different possibilities to automatically adjust visual representations should be examined thoroughly in collaboration with specialists in cognitive science. Accordingly, the visual concepts suggested in this thesis (i.e., data visualizations, implicit event representations, explicit event representations, user interfaces, and visual editors) should undergo real user tests to prove their usefulness or to reveal points for further improvements.

Finally, it can be stated that event-based visualization as introduced in this thesis is an appropriate useful basis for user-centered visual analysis. To take full advantage of incorporating event-based concepts into the world of visualization, more work remains to be done in the future (for tasks printed bold, it is suggested to collaborate with domain experts):

1. Improve event types (short-term)
2. Conduct user studies (short-term)
3. Investigate further explicit event representations (short-term)
4. **Increase event detection efficiency (mid-term)**
5. **Enhance event specification (long-term)**
6. **Analyze perceptual issues (long-term)**
7. **Develop a specification language for implicit event representation (long-term)**

Event-based visualization is an initial answer to the question asked at the beginning of this thesis *...what actually needs to be shown to intuitively draw representational conclusions* [Amar and Stasko, 2005]. Finding appropriate solutions to the discussed tasks and problems will lead to a better support for user-centered visualization. Hence, the user acceptance and the applicability of visual methods for analyzing larger volumes of data will further increase.

A. Further Conceptual Considerations

A.1. SVG, X3D, and RDBMS in the Context of Event-Based Visualization

This appendix will discuss a more technical aspect of event-based visualization. Specifically, a concept is suggested that integrates the standardized graphics formats SVG (Scalable Vector Graphics) and X3D (Extensible 3D) as well as active databases to achieve event-based visualization. The idea is to provide a tighter coupling of databases and the visualization as demanded in Section 2.2.

SVG and X3D are standardized XML-based data formats for describing and storing interactive graphics. Basically, in SVG and X3D, XML tags are used to define 2D or 3D graphical primitives. Attributes associated with the XML tags are used to describe the properties of graphical primitives. This makes SVG and X3D particularly useful for storing the output of the mapping step of the visualization pipeline [Geroimenko and Chen, 2005]. To actually display graphics stored as SVG or X3D, so called user agents interpret the XML tags and render the graphical content. Since various user agents (i.e., SVG or X3D viewers) are available for different system platforms, the stored graphics can be displayed on almost every computer, even on mobile devices [Tominski and Bieber, 2003]. This allows a distribution of interactive graphics without the need of implementing visualization techniques for different platforms. Quite the contrary, visualization techniques just generate SVG or X3D files and leave the rendering of the graphics to the user agents. For presentation purposes, it would also make sense to export achieved expressive visualization results as SVG or X3D files (e.g., a data analyst found something interesting in the data and must communicate this to a manager). A drawback of using SVG or X3D is that visualization techniques are restricted to the graphical primitives available in these formats. Moreover, due to the relinquished control over the rendering step, highly specialized visualizations that require efficient drawing are difficult to realize using SVG or X3D. Nonetheless, the possibility to store the output of the visualization mapping step in an XML-based format opens up a chance for a tighter coupling of databases and visualization as described next.

With the advent of XML-ready relational database management systems (RDBMS), it is possible to implement an event-based visualization completely within the database environment. For this, it is assumed that the RDBMS is capable of handling relational as well as XML structured data. Furthermore, the RDBMS must be an active system, i.e., must provide event-condition-action (ECA) rules. If these requirements are fulfilled, the RDBMS can store the raw data and alongside interactive graphics (as SVG or X3D) representing the data. Similar to the approaches suggested in [Díaz et al., 1994] and [Leissler et al., 2000] the graphical representations can be automatically kept

up to date using ECA rules. Additionally, the condition part of the ECA rules can be used to search the data for interesting data portions and to automatically adapt the generated visual representation to highlight these portions. Since the geometry and associated properties are also stored in the RDBMS, this can be achieved without any additional software. The advantage of this approach is that all calculations can exploit the capabilities of the RDBMS. This is particularly helpful in terms of the event mechanisms (rule execution, incremental evaluation, etc.) and the uniform access to data and graphics. Secondly, the graphical representations always show the current data and highlight relevant information. This means that if the data have changed, almost immediately an updated graphical representation can be accessed via a simple query to the RDBMS. This enables users to visually analyze data stored in a central RDBMS from any computer that has an SVG or X3D viewer installed.

Even though automatically generated visual representations based on SVG or X3D will certainly be not as complex and effective (in terms of performance) as hand-crafted visualization techniques, the tight coupling of data and visualization is a promising idea worth of investigation. An implementation of the suggested idea could hardly be realized these days. By the time of the conceptualization of the idea, no RDBMS was available that fulfilled the stated requirements. Moreover, the number of available user agents, particularly for X3D was limited.

A.2. Mobile and Event-Based Visualization

In recent years, advances in micro-electronics have led to a higher level of integration of computing devices, which nowadays show steadily increasing functionality. Smaller mobile computers, including notebooks, tablet PCs, or personal digital assistants (PDAs), have become an integral part of the world market of machines for personal computing. Although the performance of mobile devices has increased over the years, there are still some limitations that make it challenging to implement visualization concepts on such devices. That visualization of information on mobile devices is a current topic of research is underlined by very recent publications in the IEEE Computer journal [Chittaro, 2006], [Subramanya and Yi, 2006].

The following discussion will elucidate how the challenge of visualization on mobile devices, particularly on devices with small displays (e.g., PDAs or mobile phones), can be tackled and what could be the role of event-based visualization in such a scenario. The basis for this discussion is a review of current limitations of mobile devices and an exemplary description of concepts that aim at coping with these limitations. Secondly, it will be described how the concept of event-based visualization can be exploited to facilitate the visual presentation of data on mobile devices.

When representing data visually on mobile devices, it is mandatory to take the limitations of such devices into account. Although early mobile devices suffered from major limitations, today's devices have managed to narrow the gap between the mobile and the stationary world. Nonetheless, several limitations still exist and will also exist in the future. The most apparent one is the limitation of the physical display space, which varies between five and two inches. The display resolution, which varies between

240×320 and 800×600, and the color depth, which lies between 16bit and 24bit, are also somewhat limited compared to stationary devices' displays. Mobility implies another serious problem – limited power supply. Due to this limitation processors of mobile devices are not as powerful as their stationary counterparts. Further limitations of mobile devices regard the random access memory, the storage space, and the connectivity, which are not visual per se, but yet have impact on visualization on mobile devices. For a more detailed review of the capabilities of current mobile devices, the interested reader is referred to [Tominski et al., 2006b].

Despite the described limitations, mobile computers are helpful tools in many application scenarios, including maintenance support, mobile information systems, and personal digital assistance. In most of these scenarios, the communication of information plays an important role. To support this task, expressive visualization techniques are required that take the limitations of mobile devices into account. To give a concrete example without going into too much detail, the visualization of personal tasks on mobile devices will be illustrated (see [Tominski and Bieber, 2003]). Commonly, personal tasks are presented using textual lists, spreadsheets, or calendar-based representations. The visual representation at finer levels of temporal granularity is only rarely considered. On the other hand, personal tasks can be intuitively presented using a SpiraClock [Dragicevic and Huot, 2002]. Since the SpiraClock uses a clock metaphor in combination with a spiral time axis, the state of currently active tasks as well as tasks that are scheduled in the future can be easily comprehended by users. However, the SpiraClock in its original form cannot be applied on mobile devices. The computational effort required to render a spiral contradicts to the limited computational power of mobile devices (see [Taponecco and Alexa, 2002]). A possible solution to this problem is to adapt the original SpiraClock such that a less complicated shape is used for the time axis. Figure A.1 illustrates how this can be accomplished. The figure shows that the geometric complexity of the SpiraClock has been reduced to allow their application for the visualization of personal tasks on differently powerful mobile devices. This example illustrates why mobile visualization necessitates a rethinking of known techniques and the development of dedicated mobile visualization approaches.

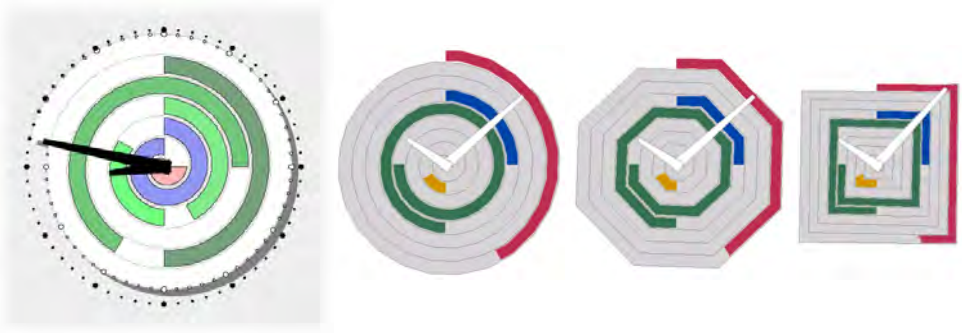


Figure A.1.: Reducing the geometric complexity of the SpiraClock.

In contrast to the previously described visualization, which is computed entirely on the mobile device, representing larger multivariate datasets (e.g., if a manager needs to assess current business data), commonly requires access to data that are not available on the device, but are stored on a server. This implies that a decision must be made which parts of the visualization pipeline should be processed on the server and what should be computed on the client. The answer to this question determines the type and the volume of data that must be transferred to the mobile client. In general, five different possibilities are applicable in such client-server-scenarios. Figure A.2 depicts all these possibilities. However, due to the described limitations of mobile devices, shifting more computations to the client side than illustrated in (a) and (b) is impractical, and hence, the possibilities (c), (d), and (e) are not applicable for mobile clients.

How visualization can potentially benefit from event-based concepts in the cases (a) and (b) is illustrated next. In fact, increasing the relevance of visual representations by focussing on events is even more important in mobile visualization scenarios than it is for stationary computers. The reason being that the number of pixels is rare and the few that are available should be used as efficiently as possible. Therefore, it should be possible for users not only to request a visualization from the server, but also to specify event types they are interested in (certainly by simple selection, see Figure 4.8 on page 62). The visualization technique should then generate either a raster image in case (a) or a vector image in case (b) that incorporates an emphasis on relevant event instances.

The advantages and disadvantages of using either raster or vector images to represent graphical contents on mobile devices have been investigated in [Tominski et al., 2006b]. These investigations have shown that both possibilities have their pros and cons. In the first case, the whole visualization pipeline is processed on the server. The server transmits a raster image to the client whose only task is to present the raster image. The advantage of this procedure is that virtually arbitrarily complex computations can be performed within the visualization pipeline. It is not necessary to take the

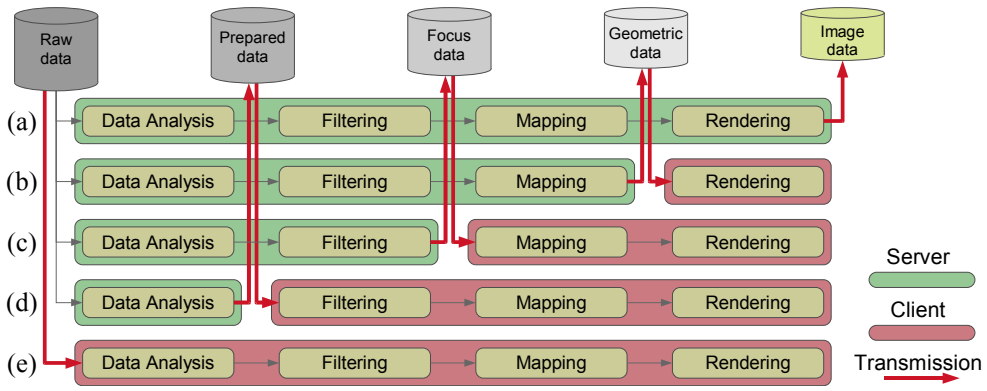


Figure A.2.: Processing the visualization pipeline in client-server scenarios.

limited computational power of mobile devices into account. However, the user cannot interact with the visual representation, except of zooming and panning the transmitted image. For case (b), the server transmits the result of the mapping step, i.e., geometric data stored in some type of vector image. The client is in charge of performing the rendering to generate the final visual representation. This enables more interactions with the visualized data and higher presentation quality.

If only raster images are transmitted between the server and the client, it makes sense to take advantage of the capabilities of JPEG2000 [Taubman and Marcellin, 2002]. Since JPEG2000 incorporates the concept of regions of interest, an emphasis on image parts that represent event instances can be easily achieved by encoding regions of interest into the JPEG2000 data stream. During the transmission of the JPEG2000 data stream, regions of interest are sent with higher priority. In other words, the most relevant visual content is transmitted first, and hence can be perceived earlier than less-relevant content. Whereas this way of exploiting JPEG2000 addresses the transmission of visualization results, it is also possible to make use of regions of interest to actually highlight event instances in the transmitted image. The JPEG2000 encoding hierarchy can be exploited for this purpose. Specifically, the parts of the image that represent event instances are visualized in full detail, whereas less-relevant parts are shown only at lower resolution. The result is a visual representation that shows relevant information in sharp detail; less-relevant information is shown only blurred. That such representations are useful not only to emphasize on event instances, but also to guide users in the image exploration process in general, is corroborated by the application of blurring in diverse scenarios (e.g., [Kosara, 2001], [Rosenbaum and Schumann, 2006]). JPEG2000 offers a further possibility that is useful for event-based visualization. In particular, it is possible to exchange parts of a JPEG2000 image without retransmitting the whole image. This is useful if the server detects an event instance after the visual representation has been delivered to the client. In this case, a highlighting of the event instance can be achieved by simply exchanging parts of the already transmitted image. All in all, exploiting JPEG2000 is a promising way of bringing event-based visualization to mobile devices.

A research direction for the future could be to achieve a similar event support for the case that vector images are transmitted from the server (see Figure A.2 (b)). Graphics standards that can be considered in this regard are SVG (Scalable Vector Graphics) and X3D (Extensible 3D). Since both of these standards are based on XML, they fit very well into client-server scenarios. Moreover, an active research community is investigating different aspects of standardized 2D and 3D graphics in such scenarios (see SVG Open at www.svgopen.org). A current hot topic is to bring progressive content transmission, which has been used in raster images for a long time, to vector graphics. If these first steps are successful, SVG and X3D can be very helpful for implementing uniform, yet client-adapted, event-based visualizations for mobile devices.

Bibliography

Abello, J. (2004). Hierarchical graph maps. *Computers & Graphics*, 28(3):345–359.

Abello, J. and Korn, J. (2002). MGV: A System for Visualizing Massive Multidigraphs. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):21–38.

Abello, J., Resende, M. G. C., and Sudarsky, S. (2002). Massive Quasi-Clique Detection. In *Proceedings of 5th Latin American Symposium on Theoretical Informatics*, pages 598–612. Springer Verlag.

Abello, J., van Ham, F., and Krishnan, N. (2006). ASK-GraphView: A Large Scale Graph Visualization System. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):669–676.

Aigner, W. (2006). *Visualizing Time and Time-Oriented Information: Challenges and Conceptual Design*. PhD thesis, Vienna University of Technology, Vienna.

Allen, J. F. (1991). Time and Time Again: The Many Ways to Represent Time. *International Journal of Intelligent Systems*, 6(4):341–355.

Allen, J. F. and Ferguson, G. (1994). Actions and Events in Interval Temporal Logic. *Journal of Logic and Computation*, 4(5):531–579.

Amar, R. A. and Stasko, J. T. (2005). Knowledge Precepts for Design and Evaluation of Information Visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):432–442.

Andrienko, N. and Andrienko, G. (2006). *Exploratory Analysis of Spatial and Temporal Data*. Springer Verlag, Berlin.

Ankerst, M., Berchtold, S., and Keim, D. A. (1998). Similarity Clustering of Dimensions for an Enhanced Visualization of Multidimensional Data. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'98)*, pages 52–60. IEEE Computer Society Press.

Atzeni, P. and de Antonellis, V. (1993). *Relational Database Theory*. Benjamin/Cummings, Redwood City.

Balkir, N. H., Ozsoyoglu, G., and Ozsoyoglu, Z. M. (2002). A Graphical Query Language: VISUAL and Its Query Processing. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):955–978.

Banks, J., Carson, J. S., Nelson, B., and Nicol, D. M. (2001). *Discrete-Event System Simulation*. Prentice Hall, Upper Saddle River.

Baralis, E. (1999). Rule Analysis. In Paton, N. W., editor, *Active Rules in Database Systems*, chapter 3, pages 51–67. Springer-Verlag, New York.

Baralis, E. and Widom, J. (1995). Using Delta Relations to Optimize Condition Evaluation in Active Databases. In *RIDS '95: Proceedings of the Second International Workshop on Rules in Database Systems*, pages 292–308. Springer-Verlag.

Bartram, L., Ware, C., and Calvert, T. (2001). Moving Icons: Detection And Distraction. In *Proceedings of 8th TC13 IFIP International Conference on Human-Computer Interaction (INTERACT'01)*. IOS Press.

Battista, G. D., Eades, P., Tamassia, R., and Tollis, I. G., editors (1999). *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, Upper Saddle River.

Bergman, L. D., Rogowitz, B. E., and Treinish, L. A. (1995). A Rule-based Tool for Assisting Colormap Selection. In *Proceedings of IEEE Visualization (Vis'95)*, pages 118–125. IEEE Computer Society.

Björk, S., Holmquist, L. E., and Redström, J. (1999). A Framework for Focus+Context Visualization. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'99)*, pages 53–56. IEEE Computer Society Press.

Bottoni, P., Costabile, M. F., Levialdi, S., and Mussio, P. (1995). Formalising Visual Languages. In *Proceedings of IEEE Symposium on Visual Languages*, pages 45–52. IEEE Press.

Bottoni, P. and Costagliola, G. (2002). On the Definition of Visual Languages and Their Editors. In *Proceedings of the 2nd International Conference on the Theory and Application of Diagrams*, pages 305–319. Springer-Verlag.

Brath, R. (1997). Metrics for Effective Information Visualization. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'97)*, pages 108–111. IEEE Computer Society Press.

Brewer, C. A. (1994). Color Use Guidelines for Mapping and Visualization. In MacEachren, A. and Taylor, D., editors, *Visualization in Modern Cartography*. Elsevier Science, Tarrytown.

Brockhaus (2003). *Brockhaus multimedial 2003 premium*. Brockhaus Verlag, Mannheim.

Brown, M. H. and Sedgewick, R. (1998). Interesting Events. In Stasko, J. T., Domingue, J., Brown, M. H., and Price, B. A., editors, *Software Visualization: Programming as a Multimedia Experience*. MIT Press.

Buja, A., McDonald, J. A., Michalak, J., and Stuetzle, W. (1991). Interactive Data Visualization using Focusing and Linking. In *Proceedings of IEEE Visualization (Vis'91)*, pages 156–163. IEEE Computer Society Press.

Card, S. K. and Mackinlay, J. D. (1997). The Structure of the Information Visualization Design Space. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'97)*, pages 92–99. IEEE Computer Society Press.

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

Carlis, J. V. and Konstan, J. A. (1998). Interactive Visualization of Serial Periodic Data. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST'98)*, pages 29–38. ACM Press.

Carlstein, T., Parkes, D., and Thrift, N., editors (1978). *Human Activity and Time Geography*. Edward Arnold Publishing, New York.

Cassandras, C. G. and Lafortune, S. (1999). *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Boston.

Catarci, T., Costabile, M. F., Levialdi, S., and Batini, C. (1997). Visual Query Systems for Databases: A Survey. *Journal of Visual Languages and Computing*, 8(2):215–260.

Chakravarty, S. and Shahar, Y. (1999). A Constraint-Based Specification of Periodic Patterns in Time-Oriented Data. In *Proceedings of Sixth International Workshop on Temporal Representation and Reasoning (TIME'99)*, pages 29–40. IEEE Computer Society Press.

Chen, H. (2004). Compound Brushing Explained. *Information Visualization*, 3(2):96–108.

Cheng, T. S. and Gadia, S. K. (2002). The Event Matching Language for Querying Temporal Data. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1119–1125.

Chi, E. H. (2000). A Taxonomy of Visualization Techniques using the Data State Reference Model. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'00)*, pages 69–75. IEEE Computer Society Press.

Chittaro, L. (2006). Visualizing Information on Mobile Devices. *IEEE Computer*, 39(3):40–45.

Chittaro, L. and Combi, C. (2001). Representation of Temporal Intervals and Relations: Information Visualization Aspects and their Evaluation. In *Proceedings of International Symposium on Temporal Representation and Reasoning*, pages 13–20. IEEE Computer Society Press.

Chittaro, L. and Combi, C. (2003). Visualizing Queries on Databases of Temporal Histories: New Metaphors and their Evaluation. *Data and Knowledge Engineering*, 44(2):239–264.

Chittaro, L., Combi, C., and Trapasso, G. (2003). Data mining on temporal data: a visual approach and its clinical application to hemodialysis. *Journal of Visual Languages and Computing*, 14(6):591–620.

Cleveland, W. S. (1993). *Visualizing Data*. Hobart Press, Summit, New Jersey.

Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387.

Coupaye, T., Roncancio, C., and Bruley, C. (1999). A Visualization Service for Event-Based Systems. In *Proc. 15èmes Journées Bases de Données Avancées (BDA'99)*, pages 181–199, Bordeaux, France.

Cousins, S. B. and Kahn, M. G. (1991). The Visual Display of Temporal Information. *Artificial Intelligence in Medicine*, 3(6):341–357.

Date, C. J. (2003). *An Introduction to Database Systems*. Addison-Wesley, Reading, Massachusetts.

Díaz, O., Jaime, A., Paton, N. W., and al Qaimari, G. (1994). Supporting dynamic displays using active rules. *ACM SIGMOD Record*, 23(1):21–26.

Demetrescu, C., Finocchi, I., and Stasko, J. T. (2002). Specifying Algorithm Visualizations: Interesting Events or State Mapping? In Diehl, S., editor, *Software Visualization*, number 2269 in Lecture Notes in Computer Science, pages 16–30. Springer-Verlag, Berlin.

Derthick, M., Roth, S. F., and Kolojejchick, J. (1997). Coordinating Declarative Queries with a Direct Manipulation Data Exploration Environment. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'97)*, pages 65–72. IEEE Computer Society Press.

Dey, D., Barron, T. M., and Storey, V. C. (1996). A complete temporal relational algebra. *The VLDB Journal*, 5:167–180.

Doleisch, H., Gasser, M., and Hauser, H. (2003). Interactive Feature Specification for Focus+Context Visualization of Complex Simulation Data. In *Proceedings of Joint Eurographics - IEEE TCVG Symposium on Visualization*, pages 239–248. Eurographics Association.

dos Santos, S. and Brodlie, K. (2004). Gaining understanding of multivariate and multidimensional data through visualization. *Computers & Graphics*, 28(3):311–325.

Dragicevic, P. and Huot, S. (2002). SpiraClock: A Continuous and Non-Intrusive Display for Upcoming Events. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'02)*, pages 604–605. ACM Press.

Ebert, D., Kukla, J., Shaw, C., Zwa, A., Soboroff, I., and Roberts, D. (1997). Automatic Shape Interpolation for Glyph-based Information Visualization. In *IEEE Visualization '97 Late Breaking Hot Topics*.

Ellis, G., Bertini, E., and Dix, A. (2005). The Sampling Lens: Making Sense of Saturated Visualisations. In *Extended Abstract Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'05)*, pages 1351–1354. ACM Press.

Elmasri, R. and Navathe, S. B. (2003). *Fundamentals of Database Systems*. Addison-Wesley, Reading, Massachusetts.

Erbacher, R. F. (2002). A Component-Based Event-Driven Interactive Visualization Software Architecture. In *Proceedings of International Symposium on Information Systems and Engineering (ISE'02)*, pages 237–243, San Diego.

Erbacher, R. F., Walker, K. L., and Frincke, D. A. (2002). Intrusion and Misuse Detection in Large-Scale Systems. *IEEE Computer Graphics and Applications*, 22(1):38–48.

Erwig, M. and Schneider, M. (2002). Spatio-Temporal Predicates. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):881–901.

Francis, B. and Fuller, M. (1996). Visualisation of event histories. *Journal of the Royal Statistical Society, Series A*, 159(2):301–308.

Frank, A. U. (1998). Different Types of "Times" in GIS. In Egenhofer, M. J. and Golledge, R. G., editors, *Spatial and Temporal Reasoning in Geographic Information Systems*. Oxford University Press, New York.

Freese, J. (2005). Untersuchung der zeitlichen und räumlichen Ausbreitung von Diagnosen anhand der Maximumanalyse. Student research project, University of Rostock, Rostock.

Galton, A. and Augusto, J. C. (2002). Two Approaches to Event Definition. In *Proceedings of the 13th International Conference on Database and Expert Systems Applications*, pages 547–556. Springer-Verlag.

Gamma, E., Johnson, R. H. R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading.

Gatalsky, P., Andrienko, N., and Andrienko, G. (2004). Interactive Analysis of Event Data Using Space-Time Cube. In *Proceedings of International Conference Information Visualisation (IV'04)*, pages 145–152. IEEE Computer Society Press.

Gehani, N. H., Jagadish, H. V., and Shmueli, O. (1992a). Composite Event Specification in Active Databases: Model & Implementation. In *Proceedings of the 18th International Conference on Very Large Data Bases (VLDB'92)*, pages 327–338. Morgan Kaufmann.

Gehani, N. H., Jagadish, H. V., and Shmueli, O. (1992b). Event Specification in an Active Object-Oriented Database. *ACM SIGMOD Record*, 21(2):81–90.

Geroimenko, V. and Chen, C. (2005). *Visualizing Information Using SVG and X3D*. Springer-Verlag, London.

Gogolla, M. (1990). A Note on Translation of SQL to Tuple Calculus. *ACM SIGMOD Record*, 19(1):18–22.

Griethe, H. (2004). Einsatz von Linsentechniken für Ikonen über interaktiven Kartendarstellungen. Master's thesis, University of Rostock, Rostock.

Haber, R. B. and McNabb, D. A. (1990). Visualization idioms: A conceptual model for scientific visualization systems. In *Visualization in Scientific Computing*, pages 74–93. IEEE Computer Society Press.

Hajnicz, E. (1996). *Time Structures: Formal Description and Algorithmic Representation*. Number 1047 in Lecture Notes in Computer Science. Springer-Verlag, Berlin.

Harris, R. L. (1999). *Information Graphics: A Comprehensive Illustrated Reference*. Management Graphics, Atlanta.

Hauser, H., Ledermann, F., and Doleisch, H. (2002). Angular Brushing of Extended Parallel Coordinates. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'02)*, pages 127–130. IEEE Computer Society Press.

Havre, S., Hetzler, E., Perrine, K., Jurrus, E., and Miller, N. (2001). Interactive Visualization of Multiple Query Results. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'01)*, pages 105–112. IEEE Computer Society Press.

Healey, C. G., Booth, K. S., and Enns, J. T. (1996). High-Speed Visual Estimation Using Preattentive Processing. *ACM Transactions on Computer-Human Interaction*, 3(2):107–135.

Hellerstein, J. L., Ma, S., and Perng, C.-S. (2002). Discovering actionable patterns in event data. *IBM Systems Journal*, 41(3):475–493.

Henriksen, K., Sporring, J., and Hornbaek, K. (2004). Virtual Trackballs Revisited. *IEEE Transactions on Visualization and Computer Graphics*, 10(2):206–216.

Herman, Melançon, G., and Marshall, M. S. (2000). Graph Visualization and Navigation in Information Visualization: A Survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43.

Heuer, A. and Saake, G. (2000). *Datenbanken - Konzepte und Sprachen*. mitp-Verlag, Bonn.

Hochheiser, H. (2003). *Interactive Graphical Querying of Time Series and Linear Sequence Data Sets*. PhD thesis, University of Maryland.

Holzhüter, C. (2005). Achsenbasierte 3D-Techniken zur Disqualifiers zeitabhängiger Daten. Student research project, University of Rostock, Rostock.

Humphrey, M. C. (2000). Creating Reusable Visualizations with the Relational Visualization Notation. In *Proceedings of IEEE Visualization (Vis'00)*, pages 53–60. IEEE Computer Society Press.

Inselberg, A. (1998). A Survey of Parallel Coordinates. In Hege, H.-C. and Polthier, K., editors, *Mathematical Visualization*, chapter 3, pages 167–179. Springer-Verlag, Berlin.

Jankun-Kelly, T. J. (2003). *Visualizing Visualization - A Model and Framework for Visualization Exploration.* PhD thesis, University of California.

Jankun-Kelly, T. J., Ma, K.-L., and Gertz, M. (2002). A Model for the Visualization Exploration Process. In *Proceedings of IEEE Visualization (Vis'02)*, pages 323–330. IEEE Computer Society Press.

Keahey, T. A. (1998). Generalized Detail in Context Problem. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'98)*, pages 44–51. IEEE Computer Society Press.

Keahey, T. A. and Robertson, E. L. (1996). Techniques for Non-Linear Magnification Transformations. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'96)*, pages 38–45. IEEE Computer Society Press.

Keim, D. A. (2002). Information Visualization and Visual Data Mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):100–107.

Keim, D. A. and Kriegel, H.-P. (1994). VisDB: Database Exploration Using Multidimensional Visualization. *IEEE Computer Graphics and Applications*, 14(5):40–49.

Knüpfer, A., Brunst, H., and Nagel, W. E. (2005). High Performance Event Trace Visualization. In *Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pages 258–263. IEEE Computer Society Press.

Knuth, D. E., Pratt, V. R., and Morris, J. H. (1977). Fast Pattern Matching in Strings. *SIAM Journal of Computing*, 6(2):323–350.

Kosara, R. (2001). *Semantic Depth of Field - Using Blur for Focus+Context Visualization.* PhD thesis, Vienna University of Technology, Vienna.

Kosara, R. and Miksch, S. (2001). Visualizing Complex Notions of Time. In *Proceedings of the Conference on Medical Informatics (MedInfo'01)*, pages 211–215.

Kranzlmüller, D. (2000). *Event Graph Analysis for Debugging Massively Parallel Programs.* PhD thesis, GUP Linz, Joh. Kepler University Linz.

Kranzlmüller, D., Grabner, S., and Volkert, J. (1996). Event Graph Visualization For Debugging Large Applications. In *Proceedings of the SIGMETRICS Symposium on Parallel and Distributed Tools*, pages 108–117. ACM Press.

Kreuseler, M. and Schumann, H. (1999). Information Visualization Using a New Focus+Context Technique in Combination with Dynamic Clustering of Information Space. In *Proceedings of ACM Workshop on New Paradigms in Information Visualization and Manipulation (NPIVM'99)*, pages 1–5. ACM Press.

Kunz, T., Black, J. P., Taylor, D. J., and Basten, T. (1997). Poet: Target-System Independent Visualizations of Complex Distributed-Application Executions. *The Computer Journal*, 40(8):499–512.

Lange, S., Nocke, T., and Schumann, H. (2006). Visualisierungsdesign - ein systematischer Überblick. In *Proceedings of Simulation and Visualization (SimVis'06)*.

Lee, D., Mao, W., Chiu, H., and Chu, W. W. (2000). TBE: A Graphical Interface for Writing Trigger Rules in Active Databases. In *Proceedings of the Fifth Working Conference on Visual Database Systems (VDB5)*, pages 367–386. Kluwer Academic Publishers.

Leissler, M., Hemmje, M., and Neuhold, E. J. (2000). Automatic Updates of Interactive Information Visualization User Interfaces Supporting Multimedia Information Retrieval through Database Triggers. In *Proceedings of the Fifth Working Conference on Visual Database Systems: Advances in Visual Information Management*, pages 341–366. Kluwer Academic Publishers.

Löffler, F. and Willert, M. (2006). Einsatzmöglichkeiten visueller Anfragesprachen zur Spezifikation von Ereignissen sowie Konzeption und Realisierung eines visuellen Editors. Student research project, University of Rostock, Rostock.

López, I. F. V., Snodgrass, R. T., and Moon, B. (2005). Spatiotemporal Aggregate Computation: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):271–286.

Lyman, P. and Varian, H. R. (2003). How Much Information. Internet Report.

Ma, J. and Knight, B. (1996). A Reified Temporal Logic. *The Computer Journal*, 39(9):800–807.

Ma, S. and Hellerstein, J. L. (1999). EventBrowser: A Flexible Tool for Scalable Analysis of Event Data. In *Proceedings of 10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'99)*, pages 285–296.

Ma, S., Hellerstein, J. L., Perng, C.-S., and Grabarnik, G. (2002). Progressive and Interactive Analysis of Event Data Using Event Miner. In *Proceedings IEEE International Conference on Data Mining (ICDM '02)*, pages 661–664. IEEE Computer Society Press.

Maier, D. (1983). *The Theory of Relational Databases*. Computer Science Press, Rockville, Maryland.

Matković, K., Hauser, H., Sainitzer, R., and Gröller, E. (2002a). Process Visualization with Levels of Detail. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'02)*, pages 67–70. IEEE Computer Society Press.

Matković, K., Hauser, H., Sainitzer, R., and Gröller, E. (2002b). Process Visualization with Levels of Detail. Technical Report 2002-16, VRVis Research Center, Vienna.

McCrickard, D. S., Catrambone, R., and Stasko, J. T. (2001). Evaluating Animation in the Periphery as a Mechanism for Maintaining Awareness. In *Proceedings of 8th TC13 IFIP International Conference on Human-Computer Interaction (INTERACT'01)*. IOS Press.

Minas, M. (2002). Concepts and Realization of a Diagram Editor Generator Based on Hypergraph Transformation. *Science of Computer Programming*, 44(2):157–180.

Moore, D. S. and McCabe, G. P. (1999). *Introduction to the Practice of Statistics*. W.H. Freeman, New York.

Muniandy, K. (2001). Visualizing time related events for Intrusion Detection. In *Late Breaking Hot Topic Proceedings of IEEE Symposium on Information Visualization (InfoVis'01)*. IEEE Computer Society Press.

Nocke, T. (to appear 2007). *Visuelles Data Mining und Visualisierungsdesign für Klimadaten*. PhD thesis, University of Rostock, Rostock.

North, C., Conklin, N., and Saini, V. (2002). Visualization Schemas for Flexible Information Visualization. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'02)*, pages 15–22. IEEE Computer Society Press.

Paton, N. W., editor (1999). *Active Rules in Database Systems*. Springer-Verlag, New York.

Paton, N. W. and Díaz, O. (1999). Active Database Systems. *ACM Computing Surveys*, 31(1):63–103.

Randell, D. A., Cui, Z., and Cohn, A. (1992). A Spatial Logic Based on Regions and Connection. In *Proceedings of 3rd International Conference on Principles of Knowledge Representation and Reasoning*, pages 165–176. Morgan Kaufmann.

Rao, R. and Card, S. K. (1994). The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus + Context Visualization for Tabular Information. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'94)*, pages 318–322. ACM Press.

Rase, W.-D. (1997). Fischauge-Projektionen als kartographische Lupen. *Salzburger Geographische Materialien*, 26(2):115–122.

Rauschenbach, U., Jeschke, S., and Schumann, H. (2000). General Rectangular Fisheye Views for 2D Graphics. In *Proceedings of Workshop on Intelligent Interactive Assistance and Mobile Computing (IMC'00)*.

Reinders, F. (2001). *Feature-Based Visualization of Time-Dependent Data*. PhD thesis, Delft University of Technology.

Reinders, F., Jacobson, M. E. D., and Post, F. H. (2000). Skeleton Graph Generation for Feature Shape Description. In *Proceedings of Joint Eurographics - IEEE TCVG Symposium on Visualization*, pages 73–82.

Reinders, F., Post, F. H., and Spoelder, H. J. (2001). Visualization of time-dependent data with feature tracking and event detection. *The Visual Computer*, 17(1):55–71.

Reinders, F., Post, F. H., and Spoelder, H. J. W. (1999). Attribute-Based Feature Tracking. In *Proceedings of Joint Eurographics - IEEE TCVG Symposium on Visualization*, pages 63–72.

Reinders, F., Spoelder, H. J. W., and Post, F. H. (1998). Experiments on the Accuracy of Feature Extraction. In *Proceedings of Eurographics Workshop on Visualization in Scientific Computing*, pages 49–58.

Ribarsky, W., Ayers, E., Eble, J., and Mukherjea, S. (1994). Glyphmaker: Creating Customized Visualizations of Complex Data. *IEEE Computer*, 27(7):57–64.

Risch, T. and Sköld, M. (1999). Monitoring Complex Rule Conditions. In Paton, N. W., editor, *Active Rules in Database Systems*. Springer-Verlag, New York.

Rößling, G. (2002). *ANIMAL-FARM: An Extensible Framework for Algorithm Visualization*. PhD thesis, University of Siegen.

Rogowitz, B. E. and Treinish, L. A. (1998). Data Visualisation: The End of the Rainbow. *IEEE Spectrum*, 35(12):52–59.

Roman, G.-C. and Cox, K. C. (1993). A Taxonomy of Program Visualization Systems. *IEEE Computer*, 26(12):11–24.

Rosario, G. E., Rundensteiner, E. A., Brown, D. C., Ward, M. O., and Huang, S. (2004). Mapping Nominal Values to Numbers for Effective Visualization. *Information Visualization*, 3(2):80–95.

Rosenbaum, R. and Schumann, H. (2006). JPEG2000-based Viewer Guidance for Mobile Image Browsing. In *Proceedings of IEEE Multimedia Modelling (MMM'06)*. IEEE Computer Society.

Sadri, M. R. (2001). *Optimization of Sequence Queries in Database Systems*. PhD thesis, University of California, Los Angeles.

Sadri, R., Zaniolo, C., Zarkesh, A., and Adibi, J. (2004). Expressing and Optimizing Sequence Queries in Database Systems. *ACM Transactions on Database Systems*, 29(2):282–318.

Schmitz, R. (2005). Effektive Abbildung nominaler Daten auf numerische Skalen. Student research project, University of Rostock, Rostock.

Schultz, R. (2002). *Konzepte und Methoden zur Nutzung von Wichtigkeitsinformationen im Prozess der Bilderzeugung*. PhD thesis, University of Rostock, Rostock.

Schultz, R. and Schumann, H. (2001). Importance Driven Rendering - Using Importance Information in the Rendering Process. In *Proceedings of Computer Graphics and Imaging (CGIM'01)*, pages 66–71.

Schumann, H. and Müller, W. (2000). *Disqualifiers - Grundlagen und allgemeine Methoden*. Springer-Verlag, Berlin.

Scott, C., Nyarko, K., Cappers, T., and Ladeji-Osias, J. (2003). Network intrusion visualization with NIVA, an intrusion detection visual and haptic analyzer. *Information Visualization*, 2(2):82–94.

Shaw, C. D., Hall, J. A., Blahut, C., Ebert, D. S., and Roberts, D. A. (1999a). Using Shape to Visualize Multivariate Data. In *Proceedings of Workshop on New Paradigms in Information Visualization and Manipulation (NPIVM'99)*, pages 17–20. ACM Press.

Shaw, C. D., Hall, J. A., Ebert, D. S., and Roberts, D. A. (1999b). Interactive Lens Visualization Techniques. In *Proceedings of IEEE Visualization (Vis'99)*, pages 155–160. IEEE Computer Society Press.

Shneiderman, B. (1983). Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer*, 16(8):57–69.

Shneiderman, B. (1996). The Eyes Have It: A Task by Data Type Taxonomy for Information Visualization. In *Proceedings of IEEE Symposium on Visual Languages (VL'96)*, pages 336–343. IEEE Computer Society Press.

Snodgrass, R. and Ahn, I. (1985). A Taxonomy of Time Databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 236–246. ACM Press.

Snodgrass, R. T. (1999). *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann, San Francisco.

Somervell, J., McCrickard, D. S., North, C., and Shukla, M. (2002). An Evaluation of Information Visualization in Attention-Limited Environments. In *Proceedings of Joint Eurographics - IEEE TCVG Symposium on Visualization*, pages 211–216. Eurographics Association.

Spence, R. (2001). *Information Visualization*. ACM Press, Addison Wesley, Harlow.

Stasko, J. T., Domingue, J., Brown, M. H., and Price, B. A., editors (1998). *Software Visualization: Programming as a Multimedia Experience*. MIT Press.

Steiner, A. (1998). *A Generalisation Approach to Temporal Data Models and their Implementations*. PhD thesis, Swiss Federal Institute of Technology, Zurich.

Stolte, C., Tang, D., and Hanrahan, P. (2002). Polaris: A System for Query, Analysis, and Visualization of Multidimensional Relational Databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65.

Stone, M., Fishkin, K., and Bier, E. (1994). The Movable Filter as a User Interface Tool. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'94)*, pages 306–312. ACM Press.

Subramanya, S. and Yi, B. K. (2006). User Interfaces for Mobile Content. *IEEE Computer*, 39(4):85–87.

Tang, D., Stolte, C., and Bosch, R. (2004). Design Choices when Architecting Visualizations. *Information Visualization*, 3(2):65–79.

Tansel, A. U. (1997). Temporal Relational Data Model. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):464–479.

Taponecco, F. and Alexa, M. (2002). Scan Converting Spirals. In *Proceedings of International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG'02)*, pages 115–120. UNION Agency.

Taubman, D. S. and Marcellin, M. W. (2002). *JPEG2000: Image Compression Fundamentals, Standards and Practice*. Kluwer Academic Publishers, Boston.

Taylor, D. J., Halim, N., Hellerstein, J. L., and Ma, S. (2000). Scalable Visualization of Event Data. In *Proceedings of 11th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'00)*, pages 47–58.

Theisel, H. and Kreuseler, M. (1998). An Enhanced Spring Model for Information Visualization. *Computer Graphics Forum*, 17(3):335–344.

Thomas, J. J. and Cook, K. A. (2005). *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE Computer Society Press.

Thompson, D., Braun, J., and Ford, R. (2004). *OpenDX - Paths to Visualization*. Visualization and imagery solutions, Inc., Missoula.

Tominski, C., Abello, J., and Schumann, H. (2003a). Interactive Poster: Axes-Based Visualizations for Time Series Data. In *Poster Compendium of IEEE Symposium on Information Visualisation (InfoVis'03)*. IEEE Computer Society Press.

Tominski, C., Abello, J., and Schumann, H. (2004). Axes-Based Visualizations with Radial Layouts. In *Proceedings of ACM Symposium on Applied Computing (SAC'04)*, pages 1242–1247. ACM Press.

Tominski, C., Abello, J., and Schumann, H. (2005a). Interactive Poster: 3D Axes-Based Visualizations for Time Series Data. In *Poster Compendium of IEEE Symposium on Information Visualisation (InfoVis'05)*. IEEE Computer Society Press.

Tominski, C., Abello, J., van Ham, F., and Schumann, H. (2006a). Fisheye Treeviews and Lenses for Graph Visualization. In *Proceedings of International Conference Information Visualisation (IV'06)*. IEEE Computer Society Press.

Tominski, C. and Bieber, G. (2003). Visualization Techniques for Personal Tasks on Mobile Computers. In *Proceedings of International Conference on Human-Computer Interaction (HCII'03)*, pages 659–663. Lawrence Erlbaum.

Tominski, C., Rosenbaum, R., and Schumann, H. (2006b). Graphical Content on Mobile Devices. In Khosrow-Pour, M., editor, *Encyclopedia of E-Commerce, E-Government, and Mobile Commerce*. Idea Group Publishing, Hershey.

Tominski, C., Schulze-Wollgast, P., and Schumann, H. (2003b). Disqualifiers zeitlicher Verläufe auf geografischen Karten. In *Kartographische Schriften, Band 7: Disqualifiers und Erschließung von Geodaten*, pages 47–57. Kirschbaum Verlag, Bonn.

Tominski, C., Schulze-Wollgast, P., and Schumann, H. (2005b). 3D Information Visualization for Time Dependent Data on Maps. In *Proceedings of International Conference Information Visualisation (IV'05)*, pages 175–181. IEEE Computer Society Press.

Tominski, C., Schulze-Wollgast, P., and Schumann, H. (2005c). Enhancing Visual Exploration by Appropriate Color Coding. In *Proceedings of International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG'05)*, pages 203–210. UNION Agency.

Tominski, C. and Schumann, H. (2004). An Event-Based Approach to Visualization. In *Proceedings of International Conference Information Visualisation (IV'04)*, pages 101–107. IEEE Computer Society Press.

Tory, M. and Möller, T. (2004). Human Factors In Visualization Research. *IEEE Transactions on Visualization and Computer Graphics*, 10(1):72–84.

Treisman, A. M. and Gelade, G. (1980). A Feature-Integration Theory of Attention. *Cognitive Psychology*, 12(4):97–136.

Tufte, E. R. (2001). *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, second edition.

Tversky, B., Morrison, J. B., and Betrancourt, M. (2002). Animation: Can It Facilitate? *International Journal of Human-Computer Studies*, 57(4):247–262.

van Walsum, T. and Post, F. H. (1994). Selective Visualization of Vector Fields. *Computer Graphics Forum*, 13(3):339–347.

van Walsum, T., Post, F. H., Silver, D., and Post, F. J. (1996). Feature Extraction and Iconic Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):111–119.

van Wijk, J. J. and Nuij, W. A. (2004). A Model for Smooth Viewing and Navigation of Large 2D Information Spaces. *IEEE Transactions on Visualization and Computer Graphics*, 10(4):447–458.

Ware, C. (2000). *Information Visualization: Perception for Design*. Morgan Kaufmann, San Francisco.

Wattenberg, M. (2002). Arc Diagrams: Visualizing Structure in Strings. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'02)*, pages 15–22. IEEE Computer Society Press.

Weber, M., Alexa, M., and Müller, W. (2001). Visualizing Time-Series on Spirals. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'01)*, pages 7–14. IEEE Computer Society Press.

Wong, P. C. and Bergeron, R. D. (1997). 30 Years of Multidimensional Multivariate Visualization. In Nielson, G. M., Hagen, H., and Müller, H., editors, *Scientific Visualization, Overviews, Methodologies, and Techniques*, chapter 1, pages 3–33. IEEE Press, Los Alamitos.

Wong, P. C. and Thomas, J. (2004). Visual Analytics. *IEEE Computer Graphics and Applications*, 24(5):20–21.

Wu, B. and Dube, K. (2001). PLAN: A Framework and Specification Language with an Event-Condition-Action (ECA) Mechanism for Clinical Test Request Protocols. In *Proceedings of 34th Annual Hawaii International Conference on System Sciences (HICSS-34)*, volume 6. IEEE Press.

Ye, N., editor (2003). *The Handbook of Data Mining*. Lawrence Erlbaum Associates, Mahwah, New Jersey.

Zhai, S., Wright, J., Selker, T., and Kelin, S.-A. (1997). Graphical Means of Directing User's Attention in the Visual Interface. In *Proceedings of 6th TC13 IFIP International Conference on Human-Computer Interaction (INTERACT'97)*. Chapman & Hall.

Zhang, R. J. and Unger, E. A. (1996). Event specification and detection. Technical Report TR CS-96-8, Kansas State University.

Zhou, M. X. and Feiner, S. K. (1998). Visual Task Characterization for Automated Visual Discourse Synthesis. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'98)*, pages 392–399. ACM Press.

Selbstständigkeitserklärung

Ich erkläre, dass ich die eingereichte Dissertation selbstständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Rostock, 09. Juni 2006 .....

Christian Tominski

Theses

1. Visual methods are helpful tools for exploring and analyzing larger volumes of data. However, the visualization methods known in literature are often tailored to particular visualization problems and do not consider the interests and needs of the users.

2. To increase effectiveness, flexibility, and relevance of visual representations, it makes sense to shift the interests and needs of users into the focus. User-centered visualization can be achieved by pursuing an event-based approach, that is, special occurrences in the data (i.e., the events) that are particularly of interest will be emphasized in the visual representation.

3. Several event-related visualization methods are known in literature. A classification schema has been developed to assess the available approaches. Even though the reviewed techniques are useful for particular visualization problems, none of them provides a general model that can be applied in arbitrary visualization scenarios.

4. To promote user-centered visual analysis, a novel generalized model of event-based visualization has been developed. The basic idea is to let users specify their interest by means of event types, to detect instances of these events in the data, and to create visual representations that are adjusted with respect to the detected event instances.

5. The developed generalized event-based visualization model comprises the three basic steps *event specification*, *event detection*, and *event representation*.

6. The task of the *event specification* is to bridge the gap between the formal world of computational methods and the informal interests that users have in their minds.

   Event formulas have been developed to enable the specification of event types with respect to tuples, attributes, and sequences of tuples within relational datasets, or to combine event types to composite event types. The proposed event formulas are based on predicate logic (PL-I), and hence allow for a broad range of interests to be specified as event types.

   To assist users with different experience in specifying their interests as event types, different levels of abstraction of the event specification formalism have been compiled into a user-centered model of event specification. This model enables users to specify event types directly, to parameterize event templates, or to simply select interesting event types from a predefined collection.

   To further aid in the event specification, visual methods are applied. In particular, a visual editor for tuple event types has been developed.

7. The *event detection* step is responsible for finding data portions that satisfy the interests specified by users. For doing so, the specified event types must be

evaluated with respect to the data at hand. To accomplish this task, methods have been developed that utilize the capabilities of relational databases and the Optimized Pattern Search (OPS) algorithm.

8. The *Event representation* step is the interface between event-based concepts and visualization methods. Two basic approaches for representing events are proposed. Events can be represented either implicitly or explicitly.

   Representing events implicitly means adapting the parameters of visualization techniques in such a way that an emphasis of the users' interests is achieved. Implicit event representations are modeled as instantaneous actions or gradual processes that are automatically applied if events are detected. Implicit event representations can be incorporated at any stage of the visualization pipeline. A variety of implicit event representations that can be useful in different application scenarios is suggested in this thesis.

   Explicit event representations focus on visualizing events, rather than on representing data. This lifts the visual analysis to a higher level of abstraction. Several visualization techniques for representing events explicitly in time and space are suggested.

9. To demonstrate the applicability of the novel generalized model of event-based visualization, the visualization framework eVis has been developed. This framework includes a TableLens, novel axes-based visualization techniques (toolkit VisAxes), and advanced map-based visual representations (toolkit TimeMap). All techniques incorporate implicit or explicit event representations to provide users with visual representations that are adapted to their tasks at hand.

10. The developed concept of event-based visualization can also be used in other application domains. This is corroborated by the integration of event-based concepts into the graph visualization tool CGV. In particular, a visual method for specifying event types with respect to graph vertices and several lens techniques for representing events implicitly is proposed.