FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Visual Assistance for Importing Time-oriented Data Tables

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Media and Human-Centered Computing

eingereicht von

Ing. Boris Serdar, BSc
Matrikelnummer 01025657

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Priv.-Doz. Dipl.-Ing. Dr. Wolfgang Aigner, MSc
Mitwirkung: Dipl.-Ing. Mag. Alexander Rind

Wien, 31. Mai 2019

Boris Serdar	Wolfgang Aigner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Erklärung zur Verfassung der Arbeit

Ing. Boris Serdar, BSc
Herbeckstraße 69/9
1180 Wien

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

Wien, 31. Mai 2019

Boris Serdar

Acknowledgements

I would first like to thank Wolfgang Aigner and Alexander Rind. Whenever I had a question about my research or writing, they found time and had always advices, which led me to the right answers.

I would also like to thank the usability experts who were involved in the usability inspections for this research project. Without their passionate participation and input, the evaluation could not have been successfully conducted and the software could not have been put on this high level.

I would also like to thank my colleagues at Syngroup Management Consulting as well as the colleagues at Syn IT Services. They always showed discernment, if it was necessary to give priority to the work on the master thesis, even if the work in the company was also of high importance.

Finally, I must express my very profound gratitude to my parents and to my girlfriend for providing me support and continuous encouragement, throughout my years of study and through the process of researching, implementation and writing this thesis. This accomplishment would not have been possible without them.

Thank you.

Abstract

There are many approaches, which were developed in the last few years, dealing with the visualization of time-oriented data. Most of the methods, used in those approaches, are very specific and made for a small number of particular analysis problems. The main reason for that is the high complexity of applications, when multiple aspects of time-oriented data are considered. However, before any visualization can be done, the data has to be imported into the software. This might be challenging, especially when the metadata for the data to import is not given. In scope of this work a prototype of an interactive visual interface is presented, which is specifying the metadata, based on the overview of the data, providing support to the user importing data from tables. The prototype is based on the software library TimeBench. This software library provides data structures and algorithms, which deal with multiple aspects of time-oriented data.

This work is giving an answer to the question: *How can a visual interface, which includes the design aspects named by Aigner, Miksch, Schumann & Tominski [Aigner et al., 2011], assist the users to import time-oriented data tables?* To answer this question a methodical approach has been applied. First of all, a literature search was performed out to see, if such a problem had already been treated in the past. User scenarios were generated to determine the scope of the work. To show how the visual interface will look like, or how the user interacts with it, mockups were created. The mockups are showing the required interaction steps of the user when creating the import configuration.

After the design of the software was determined, the interface was implemented in programming language Java. To evaluate the result, usability inspections were performed in several iterations. Those evaluations show that, although some points still exist, that can still be dealt with in future works, the software can already be used productively and the requirements set on the interface have been fully met.

Kurzfassung

Es gibt viele Ansätze, die in den letzten Jahren entwickelt wurden, um zeitbasierte Daten zu visualisieren. Die meisten Methoden, die in diesen Ansätzen verwendet werden, sind sehr spezifisch und konnten eine nur geringe Anzahl der Analyseprobleme bahandeln. Der Hauptgrund dafür ist die hohe Komplexität der Anwendungen, wenn mehrere Aspekte zeitobasierter Daten berücksichtigt werden müssen. Vor jeder Visualisierung müssen die Daten jedoch in die Visualisierungssoftware importiert werden. Dies kann eine Herausforderung sein, insbesondere dann, wenn die Metadaten für die zu importierenden Daten nicht gegeben sind. Im Rahmen dieser Arbeit wird ein Prototyp eines interaktiven visuellen Interfaces vorgestellt, welcher die Metadaten auf der Grundlage der Daten spezifiziert und den Benutzer unterstützt diese Daten aus Tabellen zu importieren. Der Prototyp basiert auf der Softwarebibliothek TimeBench. Diese Softwarebibliothek bietet Datenstrukturen und Algorithmen, die sich mit mehreren Aspekten zeitbasierter Daten befassen.

Diese Arbeit gibt eine Antwort auf folgende Frage: Wie kann ein visuelles Interface, welches die Designaspekte der zeitbasierten Daten, vorgestellt von Aigner, Miksch, Schumann & Tominski [Aigner et al., 2011] die Benutzer beim Import zeitbasierter Datentabellen unterstützen? Als Erstes wurde eine Literaturrecherche durchgeführt. Zur Festlegung des Umfangs der Arbeit wurden Benutzerszenarien definiert. Um zu zeigen, wie die visuelle Benutzeroberfläche aussehen wird, beziehungsweise wie der Benutzer mit ihr interagiert, wurden Mockups erstellt. Die Modelle zeigen die erforderlichen Interaktionsschritte des Benutzers mit der Software, die beim Erstellen der Importkonfiguration durchgeführt werden müssen.

Nachdem das Design der Software festgelegt war, wurde die Schnittstelle in der Programmiersprache Java implementiert. Um das Ergebnis zu bewerten, wurden Usability Inspections in mehreren Iterationen durchgeführt. Diese Evaluierungen zeigen, dass, obwohl noch einige Punkte existieren, die in zukünftigen Arbeiten noch behandelt werden könnten, die Software bereits produktiv eingesetzt werden kann und die ursprünglichen Anforderungen an die Software vollkommen erfüllt sind.

Contents

CHAPTER 1

Introduction

1.1 Problem Description

When designing visualizations or visual analytics applications the fundamental factor for choosing the suitable type of visualization is the data itself, that it is being designed for. For example, when visualizing the temperature variation on some specific place over some time, a line plot might be a good idea to visualize this data. Also the import of the data for that specific case might be connected with only little effort. Probably, in this case the data would be stored in some tabular shape and every row would consist of a timestamp and the temperature value. Unfortunately, the visualization of time-oriented data is mostly much more complex then in the named example. For instance, when visualizing time-oriented data which appears in periods and is repeating in specified intervals (monthly, yearly, etc.). First, the visualization of such data is a difficult process, and second, when the visualization tool for such time-oriented data is finished, the data needs to be imported into the tool which is expecting some specific "shape" of the data. One of the reasons why this task is challenging is that time has many theoretical and practical aspects, which all have to be considered when building a visual analytics applications to visualize time-oriented data [Aigner et al., 2007]. The following design aspects, hierarchical organization of time and the time elements themselves, have to be considered [Aigner et al., 2011]:

- **Scale**: In an *ordinal* time domain only relative order relations between events are described, for example: "A happened before B". An ordinal scale can be used to present such relations. Alternatively, *discrete* values describe the time values, which are mapped to a set of integers. For example, the time values are mapped approximately on the number of seconds elapsed since 1st of January 1970 in the UNIX time system.[1] In this case, the 12th of May 2000, 00:00:00 would be number

[1] For further information visit `https://www.unixtimestamp.com/`

958143982 in the UNIX time system. In contrast to the UNIX time system, where the time values between two seconds are getting approximated, the *continuous* scale allows any point between two values.

- **Scope**: In *point-based* time domain, there is no information about the region between two points in time, even if there is information for one specific point in time. In contrast, in an *interval-based* time domain, a time value stands for a specified interval. For example, a point-based time domain on the 12th of May 2000 stays for a single instant the 12th of May 2000 00:00:00 and in the interval-based time domain it may stay for the interval between the 12th of May 2000 00:00:00 and the 12th of May 2000 23:59:59.

- **Arrangement**: For people, the common perception of time is *linear*. That means, time proceeds from past to future and that each time value has an unique predecessor and successor. In a *cyclic* arrangement of time domain, a set of recurring time values appears, for example, changing, recurring values for every day of week.

- **Viewpoint**: In an *ordered* viewpoint, things happen one after the other (in a *totaly ordered* perspective only one thing can occur at a time, whereby in a *simultaneous or overlapping* perspective things can happen at the same time). A *branching* perspective can be used to present multiple scenarios. In this case, only one of the scenarios can happen. *Multiple* perspectives allow more than one view of time. In this case, the result for multiple views on the same happening might be different.

- **Granularity and calendars**: In general, granularity of time describes the mapping from time units to smaller or larger ones. For example, hours to days and days to weeks etc. A calender is a system of *multiple granularities* including the mappings between granularities. Since the granularity can be regular (1 year always includes 12 months) or irregular (one month consists of 28, 29, 30 or 31 days), the conversion between those granularities has to be supported when building time oriented applications. Beneath the support of multiple granularities, a time model might also support one, *single* granularity (every time value is specified in seconds) or *none* (for example when it has abstract measurement).

- **Time primitives**: The elements, which relate data to time (time primitives) can be divided into *instant*, *interval* and *span*. Instant is a point in time and it can have a duration (depending on granularity). In the simplest sense, intervals might be represented as a period of time between two instants. A span is not tied to a specific time, it is a primitive which represents, depending on the granularity, an amount of time, for example “one hour”.

- **Relation between time primitives**: Sometimes, the relations between time primitives might exist. For example some instants might

  – be *before*, *after* or *equal* (at the same time) to another

The relation between intervals might be the following:

  - one interval *meets* another (ends when another starts)
  - one interval *overlaps* another (one event is not finished while another starts)
  - one interval *starts* another (starts at the same time but does not have influence on the finish of the second interval)
  - one interval *finishes* another (finishes at the same time but does not have influence on the start of the secont interval)
  - one interval *equals* another (the start and finish of both intervals are equal)

- **Determinacy**: When there is complete knowledge of all temporal aspects, *determinacy* is given. In contrast to determinacy, there is indeterminacy, which is imprecise knowing of time events. It often appears in planning data with statements like: "event X will be finished in the next few days".

In most cases the time primitives does not make sense without the data which relies on the time values, or rather the context of the time values. Following the data which is connected with the time values will be discussed. There are multiple aspects on the data which also have to be considered when working with time-oriented data tables:

- **Scale**: the given data might be *quantitative*. This means that the data comparison between the data values (discrete or continuous) is possible. Besides quantitative data, *qualitative* data stands for data values which are ordered or unordered, where a metric range is not given.
- **Frame of references**: In this aspect we can distinguish between *abstract* and *spatial* data. The difference is that the abstract data does not have any relation to spatial location, while the spatial data provides this information.
- **Kind of data**: We can distinguish between *events*, which are points between *states*. A state can be seen as phase between events. For example "sitting" can be seen as a state between events "sit down" and "stand up".
- **Number of variables**: This aspect considers the number of time-dependent variables, which can be *univariate* (each time primitive is related to one single data value) and *multivariate* (each time primitive is related to multiple data values).

While discussing the connection between data and time, there are 2 important aspects: *Internal time*: when the information in the data is valid, the *external time* describes how a dataset evolves over time.

This work builds upon a library called TimeBench [Rind et al., 2013], which is able to model, visualize and process time-oriented data. TimeBench relies on design aspects

described in work from Aigner, Miksch, Müller, Schumann & Tominski [Aigner et al., 2007] and supports different time primitives like instants, intervals and spans. Also, it supports granularities, calenders and (in)determinacy. In contrast to the existing visualization tools like Improvise [Weaver, 2004], uVis[Pantazos et al., 2013], Polaris/Tableau, SAS JMP or MS Excel, which are limited according to the different design aspects already named, TimeBench provides multiple data structures and algorithms for various time-oriented data.

When performing visualization or data analysis with tools and libraries such as TimeBench, the first step is to import the data. What sounds trivial, might be challenging when the data is provided, but not its characteristics, the meta data. For example, when importing time-oriented data from some tables (i.e. CSV), one row from the table might look following:

- *Column1: "03.05.2014", Column 2: "18:10"* - since there is no further information about the data characteristics, the combination of the two columns might be a time point of temporal granularity minutes
- *Column1: "15.07.2014", Column 2: "02.08.2014"* - in this case, the presented information might be a time interval of temporal granularity days
- with no relation to the data stored in the table columns, the row number itself could be a time point of temporal granularity year starting with 2001 (based on background knowledge of the dataset or freetext documentation of the dataset - e.g. a README file)

Research Questions

According to these considerations and examples named above, I am asking the following research question:

- **How can a visual interface, which includes the design aspects named by Aigner, Miksch, Schumann & Tominski [Aigner et al., 2011], assist the users to import time-oriented data tables?**

Following hypothesis are supporting the research question:

- **[H1]: The prototyped visual interface and its usability are assisting users, with expertise in time-oriented data, to import time-oriented data tables.**
- **[H2]: The prototyped visual interface provides features based on the design aspects named by Aigner, Miksch, Schumann & Tominski [Aigner et al., 2011].**

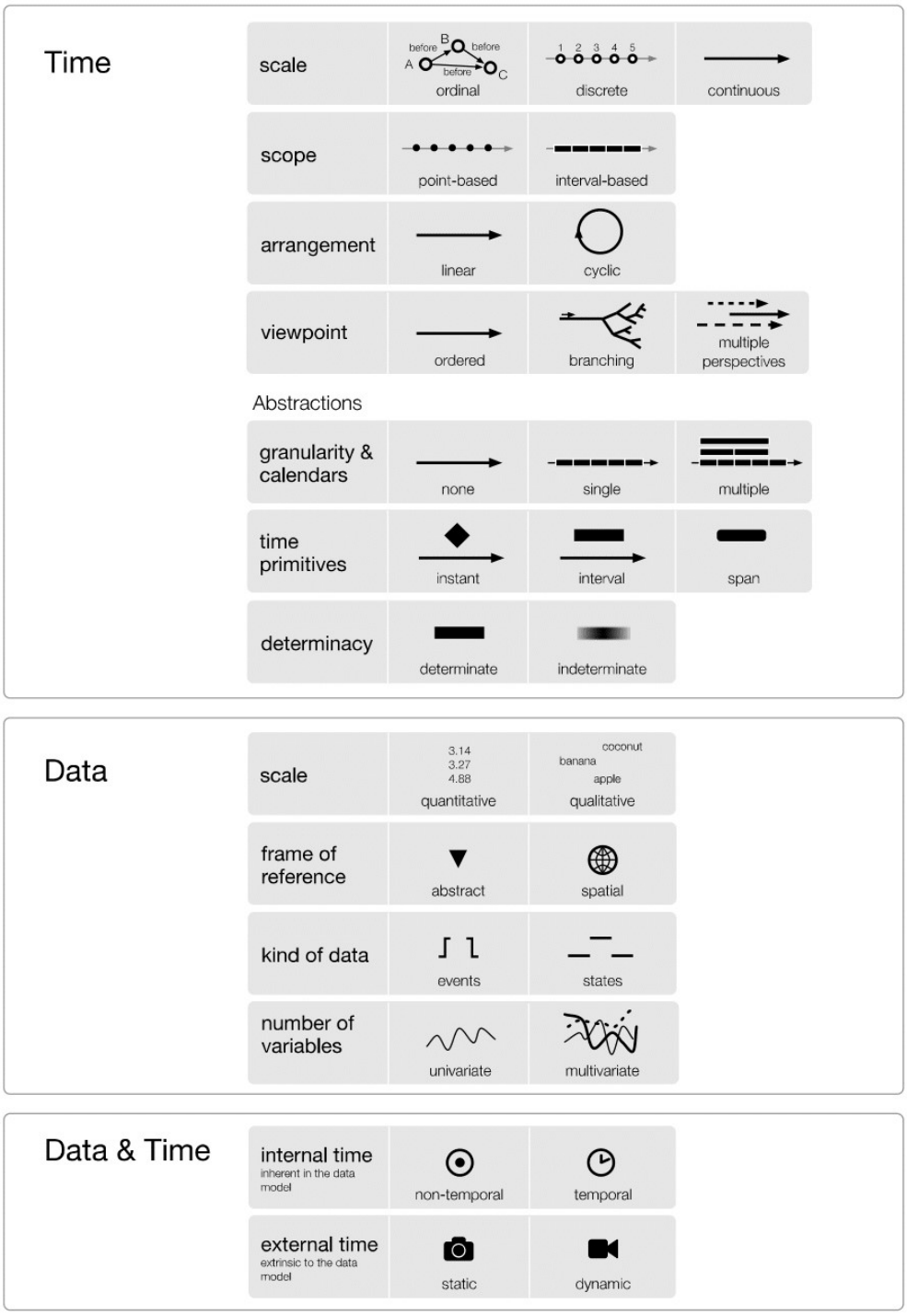


Figure 1.1: Design aspects of time-oriented data[Aigner et al., 2011]

1.2 Expected Results

The first part of the work should be a state-of-the-art literature review including projects which are standing in relation with this work. As already mentioned in Section 1.1, in the next step, an interactive visual interface for importing time-oriented data from tables will be designed and implemented as a prototype. The visual interface will be based on the software library TimeBench. The planned interactive visual interface will be:

- a validated design artefact that provides an interactive visual interface for import time-oriented data from tables,
- considering the design aspects from Aigner, Miksch, Schumann & Tominski [Aigner et al., 2011] &
- usable (avoiding usability problems set by Nielsen [1994a]).

1.3 Methodological Approach

For the first part of the work, a systematic overview of existing approaches in state-of-the-art literature will be given. To approach this, the following methods will be used:

- **Research**: Finding and description of multiple existing approaches from the literature with focus on used tools, programming languages and included functions.
- **Comparison**: Comparison of design aspects of the found approaches and highlighting of the potential approach advantages and disadvantages.

In the next step the prototype of a visual interface will be built. To achieve the expected results, named in Section 1.2, following methods will be used:

- **Creation of user scenarios**: At the beginning of the prototyping part of the work, 3 user scenarios will be created. The user scenarios will put focus on users needs and will bring the author to keeping in mind which results have to be reached during the remaining project parts. The fulfilling of the user scenarios will demonstrate the utility of the prototype in the end of the project and how the persona interacts with the system [Cooper et al., 2007] [Evans and Taylor, 2005].
- **Designing**: To visualize the goal of the work a visual interface will be designed. This step is necessary not only for author (early stage recognition of usability problems), but also for the reader, to understand and reproduce the design decisions. Using this method, the first essential errors will be minimized and the idea about the shape of the resulting project and functions will be given. To design the prototype a mock up tool will be used (i.e. Balsamiq[2]).

[2]For further information visit `https://balsamiq.com/`

- **Prototype implementation**: Using Java the interactive visual interface will be implemented.
- **Evaluation**: The created prototype will be evaluated in multiple ways. First, the software developer will test the software to avoid software errors. The evaluation and usage scenarios will show if the software prototype is fulfilling the requirement, which is to help the users import the time-oriented data. Additionally, an expert inspection [Nielsen, 1994b] will be done in more then one iteration and will be performed by usability experts. Every user scenario will be performed by the usability experts. Also, a walkthrough should be created, showing the fulfillment of the requirements, which were set by the user scenarios.

1.4 Structure of Work

The focus of the Chapter 2 is set on the related work. In detail, it will be presented first how the literature research was carried out methodically. The second part of the chapter presents the results of the literature research. These include 8 works that are related to this work.

Chapter 3 describes the design process of the visual interface. The scope of the work is defined first, then the created user scenarios are described, which specify requirements to the prototype. In addition to the 3 user scenarios, the created mockups are shown and described, which show the appearance as well as the interaction of the user with the designed prototype. Chapter 4 shows the process of implementation. In particular, the logical structure of the software is presented, and also screen shots of user interfaces are shown and described. As part of the implementation, the evolution of the design is also discussed. Chapter 5 concentrates on the conducted usability inspections and their results as part of the evaluation.

The initial requirements are compared with the results in Chapter 6. At the end of the work, a summary about the done work is given, a answer to the research question is provided and some approaches for future work are proposed.

CHAPTER 2

Related Work

2.1 Literature Research Method

Before starting the implementation, it is necessary to conduct a literature research. This should clarify whether the mentioned problem or similar problems have already been treated. There are multiple reasons for this procedure. One of them is, if exactly this problem has already been dealt with, there is probably no need for the implementation. Also, it makes sense to take up concepts, and also to learn and apply the learnt lessons from similar work .

The following search engines and digital libraries were used to find relevant literature:

- ACM DL: `dl.acm.org`
- IEEE Xplore: `ieeexplore.ieee.org`
- Springer Link: `link.springer.com`
- ScienceDirect: `sciencedirect.com`
- Web Of Knowledge: `webofknowledge.com`
- Google Scholar: `scholar.google.com`

Table 2.1 shows the list of the keywords searched for. For each library/search engine the approximate number of results, in thousand, is shown. The search provides relatively many results. The main reason for the high number of hits is the keyword *import*. Most search engines and the search algorithms in the digital libraries are not able to differentiate the search keyword *import* from the keyword *important*, which is why it is not possible to get the search results only for the exact keyword *import*.

Keywords	Google Scholar	ScienceDirect	Springer Link	IEEExplore Digital Library	ACM Digital Library	Web Of Knowledge
import	3300	360	430	4	54	1
data table import	1.000	200	200	0,05	200	1,8
intelligent import	300	14	28	0,3	120	0,8
import assistant	400	17	30	0,02	68	0,6
time based data import	1.000	200	240	0,2	330	3,4
predict datatypes	22	0,6	0,6	0	25	0,01
import time data	2.000	200	260	0,4	260	9
import optimization	200	70	37	0,4	100	2
import improvement	1.000	100	127	0,1	115	4
time based data	6.000	5.000	4.000	142	300	1.100
time oriented data	4.000	700	700	8	250	32
import time	2.500	300	370	1	154	30
data import	2.500	250	270	1,5	150	52
CSV import	40	1,7	4	0,01	50	0
support import	2.000	200	300	0,6	120	0,4
data descriptors	2.000	100	90	4,4	170	40
tabular data	300	50	43	0,5	160	9,5

Table 2.1: Approximate number of results for search engines (in thousand)

However, the keyword *important* yield a lot of results, which could not be used for this work and made finding relevant content difficult.

The literature research resulted in a collection of 39 papers (relevant after reading the abstract, some results and conclusion). In the next iteration, the found papers were read completely in order to decide, which paper is going to be named in this work. The main selection criterion was the relevance of the content of the paper. More specifically, as relevant papers were all papers selected, which were dealing with any kind of data visualization and import of any kind of data. Another criterion was the year of publication. The older a found paper was, the higher was the likelihood that the used frameworks and libraries were already obsolete. As a result of our literature research 8 relevant papers

were found and will be presented in this work.

2.2 Results of the Litearature Research

In this section, 8 approaches in this area will be presented and described. At the beginning a paper is presented, which deals with the data descriptors. With the described methods the context of the relevant data can be described. The next presented work shows how a system can automatize a number of decisions, which help visualization beginners to work with data. In contrast to this work, Excel Massive Data Intelligent Import System presents an import system, which is working with Excel and works in concert with the user input. The reason for this is to show how a piece of software can be realized as an extension to an already existing system and how the user can be assisted during the importing process. Polaris (Tableau), iVisDesigner and Lyra are very popular visualization environments. With their presentation, we would like to explain how the visualization environments present data these days and how the import is realized in those software applications. The last two works present systems, that can be used explicit as data import systems. Both use different strategies and interactions to complete their tasks.

2.2.1 A systematic view on data descriptors for the visual analysis of tabular data

So called "data descriptors" [Schulz et al., 2017] are trying to describe the data context and content. Because tabular data is the common form for structured data, the work focuses on tabular data. In this paper, the authors classify the data descriptors in 2 groups: Data Flow Descriptors (DFD) and Data Space Descriptors (DSD). Data flow is the look of the data from a temporal perspective (the data's past, the data's present and the data's future) and the data space is the look from a structural perspective. Data Flow Descriptors describe where the data comes from, where it is now and where it might go (the purpose of the data). Data Space Descriptors describe data context (granularity or dimensionality) and data content (the actual data values). The authors named 3 sources of data descriptors:

- *querying* the descriptor from additional data source (i.e. annotations)
- *deriving* the descriptor by computation from dataset and
- *user input*

Figure 2.1 shows one possible process of gathering data descriptors from tabular data. The complete process can not be done autonomously without including the user in the procedure, if the results should be correct. The first processing step describes the data type for each column. This step has to be done semi-automatically and requires user input. The reason is that the software is not able to recognize ordinal data types. After

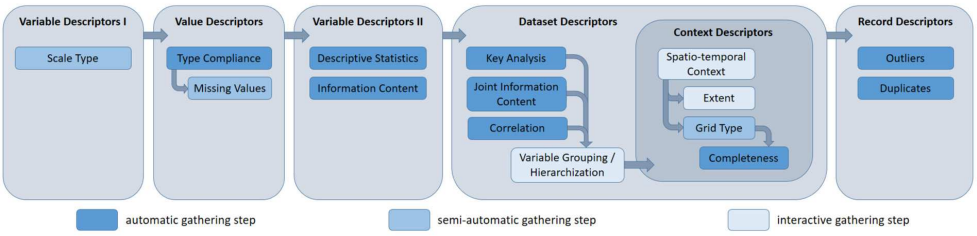


Figure 2.1: Gathering descriptors from tabular data[Schulz et al., 2017]

the completion of variable description, value description can be performed. This step describes the actual cell values and also the user might complement the missing values. Afterwards, the variable descriptor 2 operates. This is a procedure, which is gathering descriptive statistics out of the values. The fourth step provides information between data context and data content. The user is also invited to group values, for example two variables "date" and "time" to one variable "datetime". Using the context descriptors the user is able to specify the context variables which describes spatial/temporal reference and its extent (point, local and global). An automatic completeness check is possible if the descriptors are configured well and the data is suitable. The last step of the procedure is automatic recording of outliers and duplicates, based on the given data descriptors.

2.2.2 Keshif: Rapid and Expressive Tabular Data Exploration for Novices

Before any valuable and highly interactive visualizations and infographics can be created, most of Visualization design environments require the user to make many decisions. Not only is this procedure not very efficient, it also requires a lot of experience, knowledge and cognitive effort. To minimize this number of decisions, Yalcun et al. have presented a tool called Keshif [Yalcun et al., 2018]. The named tool automatically tries to extract certain insights from the data set and thus bring the user to the desired visualization as quickly as possible. Especially the novice visualization analysts benefit from Keshif, because they can focus on the data-driven insights and less with visualization specifications and underlying decisions.

Keshif has an automatic aggregation function, which depends on the data type. The tool extracts an attribute value from records and is able to aggregate records by that extracted value. Figure 2.2 shows different aggregation possibilities for each included data type. For example, data type *time* can be visualized as a line in an absolute scale or a part-of scale. Also, the aggregate metric function computes and shows valuable information about the record (statistic data like median, percentile and count).

Sometimes, the data that is imported into visualization systems, needs to be adjusted before the visualization takes place. For that purpose Keshif provides functions that return calculated data from the raw data, which is one of the main reasons, why this tool is named in this work. The calculated attributes can:

- parse token text values (e.g. split “A;B;C” on “;” to create an array of characters),
- process data without making any changes to the original data source (e.g. create numerical data types, like *10000*, from string value *10k*)
- extract time features from time data type (e.g. getting days of week from time data type)
- combine multiple record features
- combine data from multiple data sources

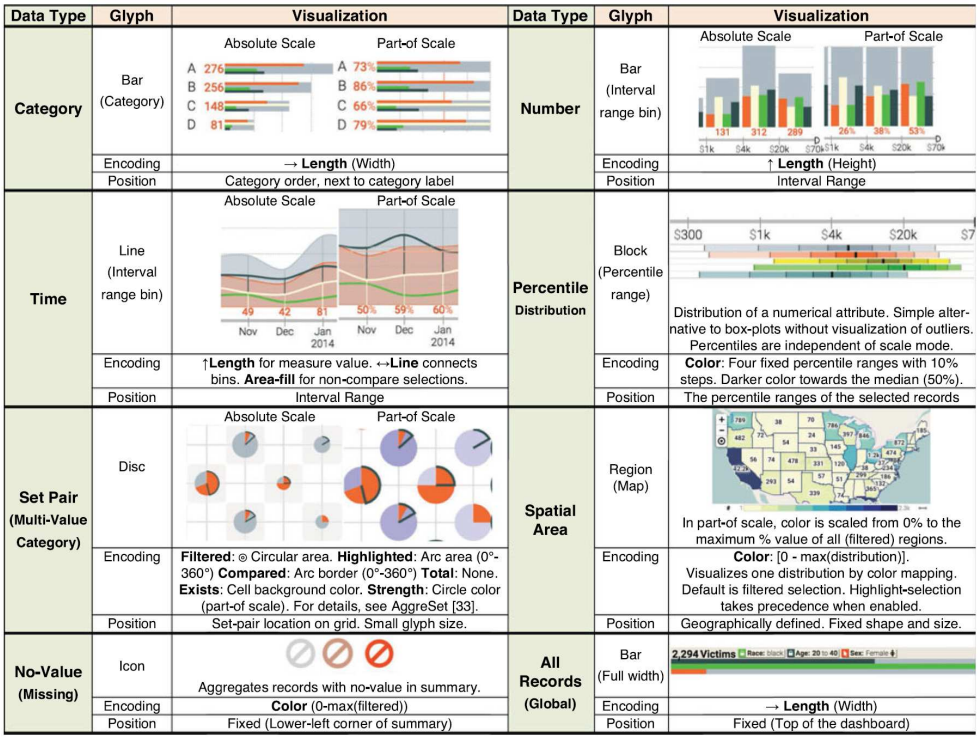


Figure 2.2: Visual Aggregate Encodings for Common Data Types [Yalcun et al., 2018]

The prototype Keshif has been created using JavaScript, HTML and CSS. To assist in the generation of web-based interactive data visualizations, the D3 library was used. The created implementation supports configuration and customizing using web programming.

2.2.3 Excel Massive Data Intelligent Import System

Excel massive data intelligent import system [Ying et al., 2010] is a software which has been implemented in order to make the import of a large amount of data into a database easier. The software was implemented for a specific purpose. It was developed to import

large amounts of data into a National Oil/Gas Resource database from Excel. The system was created in Visual Studio 2008 development tool and is using C# as developing language. The Model-View-Controller framework (MVC) has been used to separate application input, processing and output. The data is imported into an Oracle database. The system is using extract, transform and load design patterns (ETL). To extract the Excel data from the data source, the developers are using ADO.NET connection technology. To analyse and design the system, developers have used UML.

The system operates in 3 main steps: collection of data out of Excel files, processing of the found data and output of the data for Oracle database. Figure 2.3 is presenting the system workflow chart. There are multiple steps needed to import the data from an Excel spreadsheet into the Oracle database. First, the Winform interface allows the user to select the Excel tables to be imported. In the next step the system extracts the names of subjects from metadata tables and the user has to select the imported subjects and the target table to import into. After choosing the destination table, the user must choose the primary and foreign keys and also the Null values. The system uses its own data checking function to detect errors in primary keys, foreign keys and Null values. If errors are found, they are displayed in a list with the information in which row and column they occurred. If no errors are found, primary keys are created in the destination table and the data is imported.

2.2.4 Polaris (Tableau)

The idea behind Polaris [Stolte et al., 2008] is the exploration of multidimensional relational databases in an interactive way. To generate a graphic, the data can be imported from different sources. Each source gets mapped to its own *layer* in the software and can be combined with other imported sources. To specify the table configurations, the creators of Polaris defined a special algebra, which allows the execution of the cross, nest and concentration operators on itself. After the configuration, the user is asked to select the predefined type of chart or to specify the individual components of the graphic. The space of graphics is structured into groups by the type of fields mapped on the axes: ordinal-ordinal (for example a table showing sales by state and product), ordinal-quantitative (for example bar charts used to compare several functions of the independent variables) and quantitative-quantitative (for example the map explaining relationships between quantitative variables). The visual specifications are generating queries for the database. The queries prepare the dataset for the visualization in 3 steps:

- selection of the needed dataset,
- filtering, grouping and sorting of dataset into labels/panes and
- grouping, sorting and aggregating (summation, building an average) of dataset.

Figure 2.4 shows the Tableau user interface. Nearly all objects and tools can be applied to the visualization by drag and drop. The underlying data, that has been imported earlier,

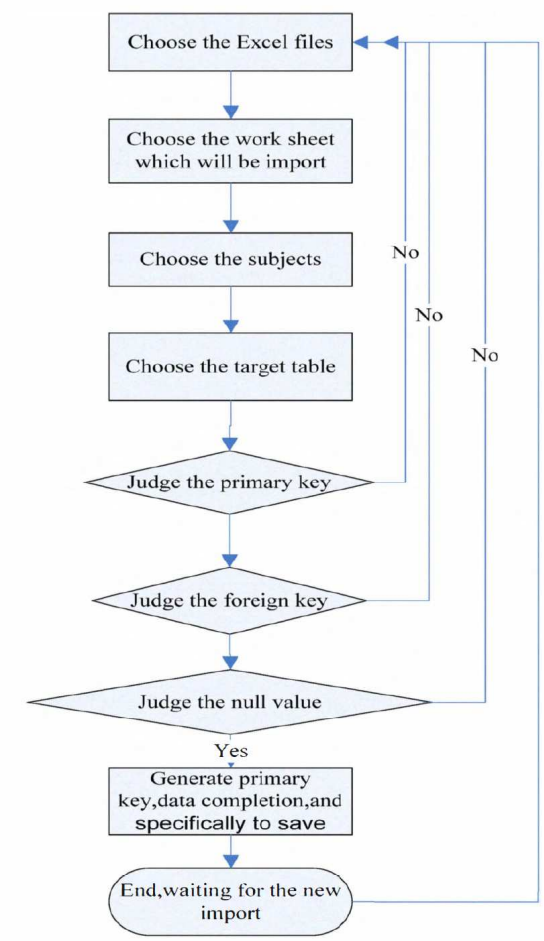


Figure 2.3: The workflow of the import system [Ying et al., 2010]

can be selected for the visualization by clicking on the wanted data table column in the tab "Data" and dragging in into the "Columns" or "Rows" field over the visualization panel. By clicking on the selected columns or rows, the user has the possibility to choose different granularity of the data. To apply the type of the visualization, the type has to be dragged from the model section and dropped into the visualization panel. Also trends and calculations can be done easily by selecting the data on the visualization panel and clicking on the correct "Model" function on the left side of the screen.

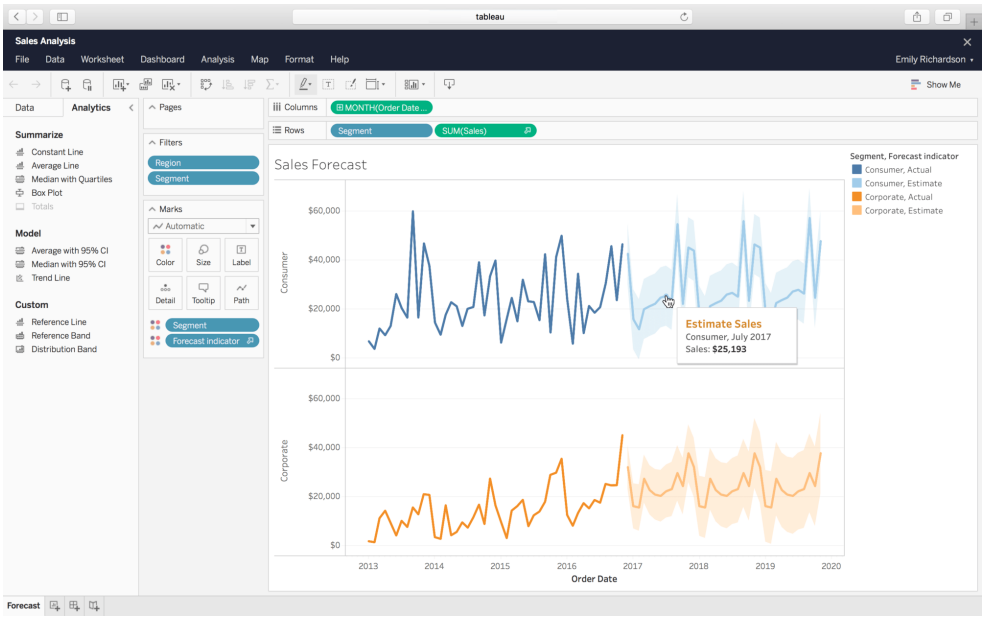


Figure 2.4: The Tableau user interface [Tableau, 2019]

2.2.5 iVisDesigner

iVisDesigner [Ren et al., 2014] is an information visualization system, which in contrast to programming frameworks for visualization like Prefuse [Heer et al., 2005] or D3.js [Bostock et al., 2011], does not require textual programming. The system is a web-based platform, which provides an user interface in a web browser and allows the designing of the visualizations by user interactions like drag and drop, sketching and by clicking on elements in the context menu. According to the input procedure, the datasets are loaded as JSON objects into the system. The import supports data serialization language YAML[1] and CSV files. While trying the online designer[2], first, we uploaded a CSV file. The system does not expect any user interaction at this point. To create a timeline visualization, it is necessary to map the CSV table rows to the given axes. The data type of the axis points is automatically "num" and the user has no influence on this option. After the data import, there was no possibility to change the datatype. We also have not found any option to influence the granularity of the data or import different time primitives, like intervals or spans.

Figure 2.5 shows the iVisDesigner interface. Similar to the Polaris interface, the panel on the left border represents the structure of dataset. The tools panel, which is placed in the upper section of the interface, provides the tools to move objects, to create them and to change the view. The biggest part of the interface consists of the canvas to draw the visualizations. In the shown example, there are multiple linked views on the same data,

[1]For further information about YAML visit `http://yaml.org/`
[2]iVisDesigner online toolkit `https://donghaoren.org/ivisdesigner/toolkit`

visualizing the air pollution in Beijing. To create the visualization, the objects have to be dragged on the canvas and combined by selection and adjustment in the object panel.

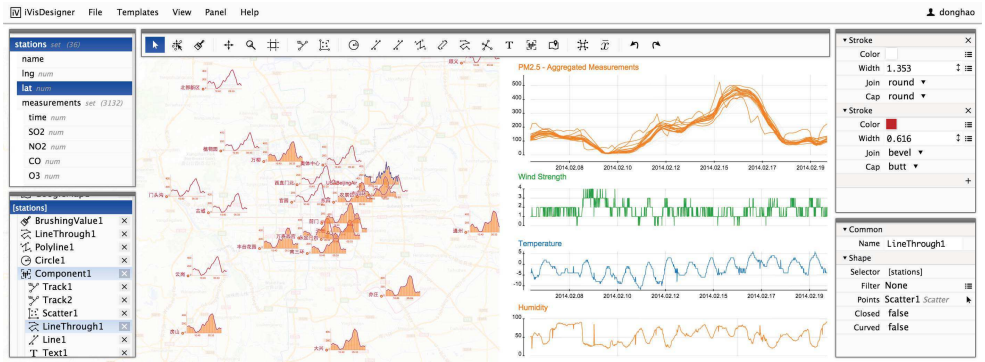


Figure 2.5: The iVisDesigner user interface [Ren et al., 2014]

2.2.6 Lyra

Similar to iVisDesigner (Section 2.2.5), Lyra [Satyanarayan and Heer, 2014] is an environment for designing and creating information visualization. It also does not require programming skills, instead it builds on common user interactions like drag and drop and clicking. After importing the dataset to the environment, the user is able to create marks, which have to be linked to the data fields. Data transformations and layout algorithms allow the creation of visualizations.

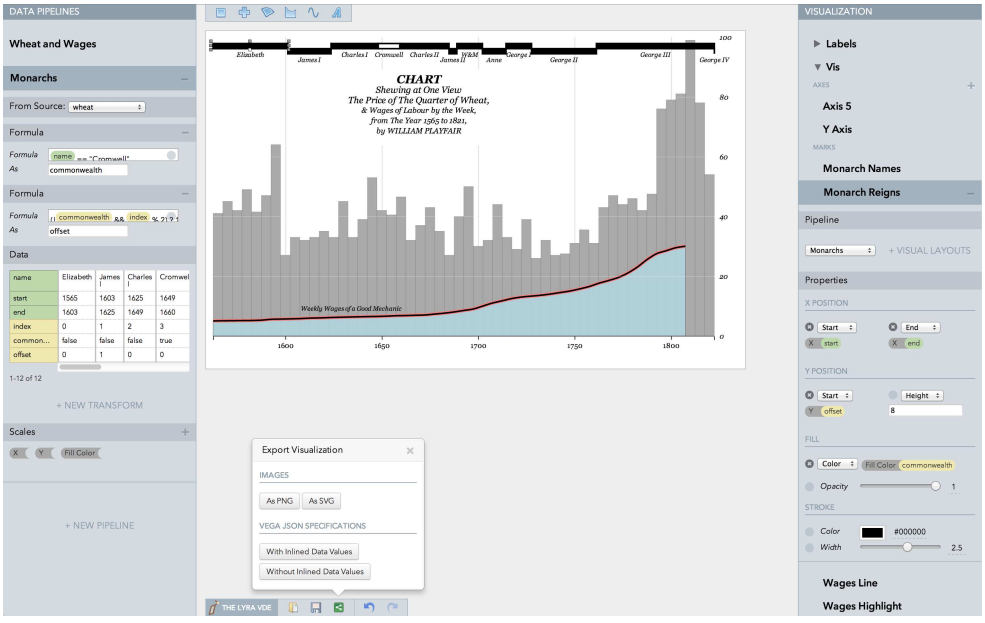


Figure 2.6: The Lyra user interface [Satyanarayan and Heer, 2014]

Figure 2.6 presents the Lyra interface. The left side-panel is presenting the data pipelines. The shown data table is demonstrating the current output and below, there is the possibility for the user to scale transforms over the given fields in the pipeline. The right side-panel is showing the visual properties. The main area for the visualization is taking place in the middle of the screen. The shown example reproduces a classical chart by Playfair.

There are 3 ways of importing the data into the online toolkit[3]: JSON, CSV and TSV (Tab Separated Values). After importing the row values to the interface, the data type for each table column has to be selected by the user. There are only 4 options for the data types: *string*, *number*, *boolean* and *date*. The creation of a simple timeline seems to be created relatively fast, however, there is no opportunity to import other time primitives (intervals, spans) or influence on the granularity, related to iVisDesigner.

2.2.7 Wrangler: Interactive Visual Specification of Data Transformation Scripts

Before any kind of visualization can be started, the data must be loaded into the visualization software. This data is not always in good condition and must be cleared due to misspellings, missing data, outliers or duplicate data. Wrangler [Kandel et al., 2011] was designed to transform raw data into a form that visualization tools can import and represent correctly. Using this tool, the analysts are able to specify transformations by executing multiple basic transforms one after another.

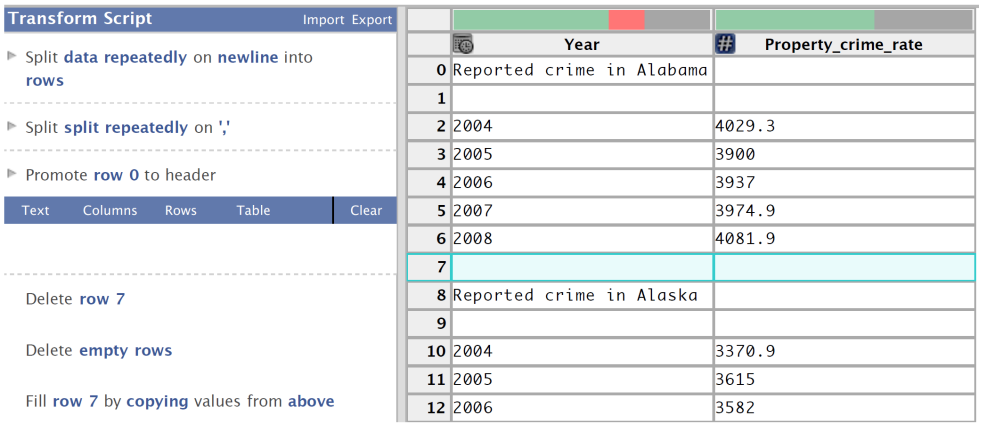


Figure 2.7: The Wrangler interface [Kandel et al., 2011]

As far as the interface is concerned, Wrangler uses a declarative transformation language which has been extended by authors with additional operators. There are 8 classes of transforms, which the Wrangler language includes:

[3]Lyra online toolkit `http://idl.cs.washington.edu/projects/lyra/app/`

- *Map* allows transforms of input data to one (extracting, cutting, splitting values into multiple columns, reformatting, arithmetic and updates), zero (deleting) or multiple output row (splitting data into multiple rows).

- *Lookups* allow combining data from external data tables (e.g. mapping zip codes to state names). Also there are 2 different types of *joins*: equi-joins and approximate joins. Are sometimes usable for correcting typos and recognizing the correct data type afterwards

- *Reshape* transform allows 2 different operators: fold (collapses multiple columns to minimum 2 columns) and unfold (creation of new column headers from values).

- Fill and lag operations are part of *positional* transforms. Fill generates values (e.g. filling empty cells with some values based on neighbour data) and lag operator moves the value up or down the column.

- *Aggregation* functions like min, max, sum, mean etc. are also provided.

- Wrangler also supports *sorting* of rows.

- *Key generation* (skolemization)

- To set column names and specify column data types *schema transforms* are also possible.

Wrangler not only supports standard data types (e.g. integers) but also semantic roles (e.g. currencies) in order to validate the data. For example, Wrangler is able to check and validate a zip code of the USA. In this case, a zip code is only correct, if the value consists of 5 integers. There are few semantic roles which are supported (geographic locations, government codes, currencies, and dates). Also, semantic roles can be extended.

Figure 2.7 shows the Wrangler interface. In the left upper panel the history of transform scripts is placed. Also, there is the possibility to import already created transform script or to export the current script. Below the transform script panel, a transform selection menu and automatic transform suggestions, depending on selected items in the right panel, are shown. The right panel shows an interactive data table and the data quality meter. Based on the user input, semantic role or data type Wrangler suggests transforms to the user. In order to make the data transforms easier, the named suggestions are using natural language descriptions and visual transform previews. By clicking on the suggested transforms, the user gets the opportunity to change the parameters of the transformation. The visual transform previews allow the user to see what modifications will take place to the data as soon as the current script gets executed.

2.2.8 Supporting heterogeneous data import for data visualization

The most common visualization tools use so-called *general array importer* functions [Ford et al., 1995]. Those functions are created to allow user to map the input data to the points or spaces in the target space (mostly N-dimensional grid). They work well for regular grids and uniform datasets. However, because of the non-uniform data files and/or non-regular grids, sometimes it is impossible to create appropriate descriptions of the imported dataset in the visualization tools. Mostly, the only help is to use a *format converter*, which changes the original dataset into a new shape, which is better accepted by the visualization tool. In addition, the question arises: how can one merge data from multiple sources into one consistent data model?

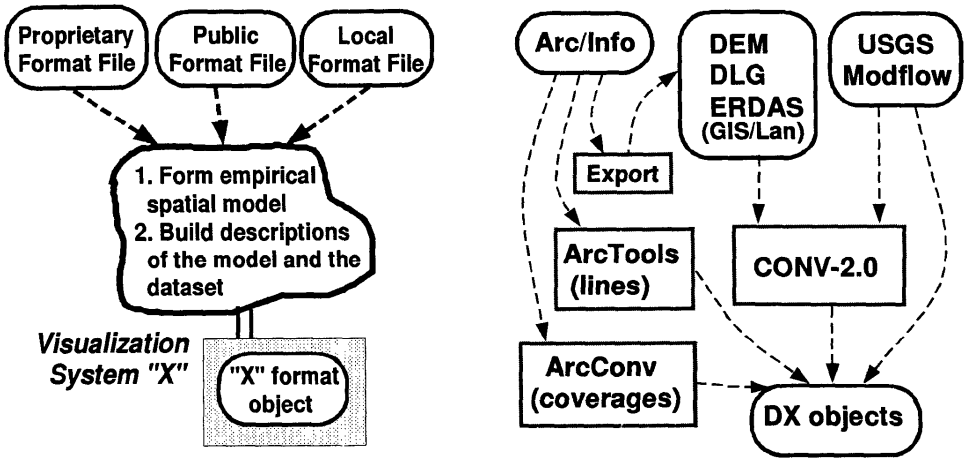


Figure 2.8: Data import problem (left) and schematic workflow of the conversion program (right) [Ford et al., 1995]

Figure 2.8 (left) is showing the different data format categories, which the import should support:

- *Proprietary Format File* is the individual format that depends on the used visualization software
- *Public Format File* is the format which is based on published standards
- *Local Format File* is based on dataset which is produced by locally written code

The point of interest for the authors was to import data into IBM - Visualization Group's Data Explorer (DX). DX supports meta-descriptions, which can be inserted into an external file. This feature allows external objects to be structured off-line and then be imported into the tool. The visualization tool also provides a programming interface and process-control abilities. These features allow execution of additional data generators and import scripts, which have been used by authors to support all the different data

formats. Figure 2.8 (right) shows the structure of the created importer. The created package CONV-2.0 can be used to send the output to a file or to put the output directly into DX. It combines converters for public formats and some chosen local formats. For the proprietary formats, the import starts with the export process. "Export" exports data into ERDAS/Lan format, which can be imported into CONV 2.0. CONV 2.0 exports then files to DX. "ArcTools" uses a macro language to export data to DX objects. "ArcConv" uses ArcInfo software library to export data to DX objects. The authors are describing 3 ways to import data to locally defined formats:

- use of external conversion,
- modify the code of the export system to export a format, which can be imported into the target visualization system and
- modify the generator process, which then can be executed in visualization environment

The workflow of the system is the following. First, the user has to select the type of the file to be converted and afterwards the system shows a dialog to the user. The dialog is showing information about the converted file and allows the user to set multiple configuration to the export, like use of standard output, setting of file names, creation of binary files or ASCII, or which bands to convert. Depending on supported data formats, the dialogs are slightly different from another. In the last step, the tool creates the chosen output using the settings, which were applied by the user.

2.3 Related Work Discussion

The work from Schulz, Nocke, Heitzler & Schumann [Schulz et al., 2017] shows their own classification of data descriptors. They do not present their own taxonomy, they present their own view on the taxonomies that already exist. In this paper the authors were concentrating on climate data, which is useful in showing detailed view on this topic. However, for my purpose it would be interesting to get insights about dealing with time and time-oriented data. For this work, it would be interesting to see, which descriptors particularly the authors recommend and how the structure of the descriptors would look like. Like the authors mention in their work, providing the data descriptors to the analyst is only one step in the visualization process. The actual usage is dependent on the current visualization task and might be more or less necessary.

With Keshif [Yalcun et al., 2018], the authors have created a tool that should minimize the number of decisions, which appear when the user is creating visualizations. The software automatically tries to get insights out of the data to facilitate the visualization analyst's work, which might be very useful. It also gives the possibility to prepare the data before processing, e.g. aggregation functionalities for included data types. The fact, that the tool uses up-to-date frameworks and libraries has a positive impact on the

applicability and usability of the tool. The fact that the tool is dedicated primarily to newcomers lacks the potential for advanced analysts. In particular, there is a lack of possibilities that would allow the user to import more complex data (types) into the tool, such as cyclic data. Likewise, it is not shown how the automatic algorithms can be avoided to extract findings manually and eventually with more insights into the data.

With Excel Massive Data Intelligent Import System [Ying et al., 2010], the authors have presented a tool that imports the data from Excel into Oracle database. The focus in this project is the interaction of the system with the user. The system requires step-by-step user decisions describing the desired import, which is the main reason why this project has been mentioned in this work. After the user has applied all keys, judged Null values, source- and target tables the import gets started. In our opinion, it would be helpful, if in these steps the system could make suggestions based on the data to be imported. In addition, the imported data types were not discussed in a appropriate level of details. It would be interesting to know how the system handles compex data types, e.g. datetime.

Project Polaris, which is now named Tableau[4], was created for exploration and analysis purposes of multi-dimensional databases. Since the system uses database queries to generate data subsets out of the original data, it might be quite comfortable for some users to import the exact data, which they require for their visualizations. Also when speaking about the importation of time-oriented data, it might be a comfortable way to import the data in the shape the analyst wants, since the visual specification is creating SQL-Queries for the database [Stolte et al., 2008] .

Different from Polaris, in other visualization tools like iVisDesigner[Ren et al., 2014] or Lyra[Satyanarayan and Heer, 2014] the user imports the data from datatypes like CSV and has less influence on the imported datatypes. In the field of time-oriented data, only the default primitives (instants) are considered. Data primitives like spans and intervals were not included.

Wrangler [Kandel et al., 2011] was created to transform raw data into a form that visualization tools can import and represent correctly. It uses natural language descriptions and visual transform previews, which help the user to create correct transforms on the raw data. The additional features, that Wrangler offers make it possible to perform more complex operations on the raw data, like mapping, lookups, reshaping, aggregations, sorting, etc. The tools also gives the opportunity to work with different semantic roles like currencies and dates. Although the tool has useful functions that can adjust raw data, it is not an import tool. The idea with the visual transform previews seems to be very helpful and will be considered in the further implementation of the work. Like the preview function, we find the history function very useful and we will keep it in our minds during the further work.

Supporting heterogeneous data import for data visualization [Ford et al., 1995] describes the work that deals with the import of data from multiple sources into one consistent data

[4]For further information visit `https://www.tableau.com/`

Time-oriented data aspect	Keshif	EMDIIS (Sec. 2.2.3)	Polaris (Tableau)	iVisDesigner	Lyra	Wrangler	DX importer
Arrangement - linear	✓	×	✓	✓	✓	×	×
Arrangement - cyclic	×	×	✓	×	×	×	×
Single granularity	✓	×	✓	×	✓	×	×
Multiple granularity	✓	×	✓	×	×	×	×
Time primitive - instant	✓	×	✓	✓	✓	×	×
Time primitive - interval	✓	×	✓	×	×	×	×
Time primitive - span	×	×	×	×	×	×	×
Determinacy	✓	×	✓	✓	✓	×	×
Indeterminacy	×	×	×	×	×	×	×

Table 2.2: Cross-sectional analysis of shown approaches and supported time-oriented aspects

model. The authors have focused on importing data into IBM - Visualization Group's Data Explorer (DX), which supports meta-descriptions (like TimeBench [Rind et al., 2013]) as external files. The created tool allows execution of additional data generators and import scripts. The described tool seems to be useful for the purpose, but however, it does not support visual assistance of the import procedure. Also, it is focused on DX, which is not the issue of this work. The authors do not mention any kind of import of time-oriented data, which would be useful for our case.

Table 2.2 represents a cross-sectional analysis for the shown approaches. It is shown whether an aspect of time-oriented data is supported by the approach or not. By "supported", it means that the application is able to visualize and/or import the data, which is underlying time-oriented data aspects.

CHAPTER 3

Design of Interactive Visual Interface

In order to determine the scope of the work, user scenarios were described in the first step. To show how users interact with the software, mockups were created in the next step. The mockups show how the user uses the interactive user interface to handle tasks set in the user scenarios. In this chapter, 3 scenarios are described, which demonstrate the need and the usage of the visual interface. The first scenario [Ilse Arlt Institut für Soziale Inklusionsforschung, 2019] is based on software from the real world, while the other scenarios are fictitious. After the user scenarios, the mentioned mockups are demonstrated and described.

3.1 The Scope

Before the user scenarios were generated, some basic questions about the software should be answered. As an example of these basic questions was the question of the software's outcome. Since the import assistant should be used as a library in external software frameworks, we agreed that the output of the import assistant should be TemporalDataset. TemporalDataset provides a data structure, able to store temporal objects and the relationships between them [Rind et al., 2013]. These TemporalObjects do have multiple roles. First, they consist of TemporalElements, which store temporal aspects of the data, and second, they refer to the non-temporal aspects of the data.

To appropriately test the software, the work requires an input and output format. These formats should be independent of other software libraries and should be simple, so that the resources can be focused on the core of the work. Based on these considerations, GraphML[1] should be used as export interface. Another question, that we had to discuss

[1] http://graphml.graphdrawing.org/

was: Which files should be supported by the import assistant? Since iCalender has already been specified by TimeBench, the prototype should support import of tabular data. As already discussed, due to the focus on the core of the work, the import of the data should be realised in a simple way. A simple interface solution, that can store tabular data, is to import data from CSV files. For the future work, also the data from other storages for tabular data, like Excel/OpenDocument, JSON and databases should also be able to get imported.

It should also be clarified, if data wrangling [Kandel et al., 2011] should be performed on the raw data, before import into the software. We agreed that the data wrangling is not the main issue of the work, however, there should be some functionality, which gives the option to the user to bring the raw data into a valid state, which afterwards can be imported into the software.

3.2 User Scenarios

There are multiple purposes for user scenarios in software development context. One of them is to develop ideas about uses and the possible ways how the user interacts with the software. Another purpose in this case is to support the dialog between software developers and people who are responsible for evaluation [Evans and Taylor, 2005].

3.2.1 User Scenario 1 - EasyBiograph

This user scenario was chosen and involved in the work for several reasons. On the one hand, the user scenario should show that the software is based on everyday situations and is not based purely on fictitious tasks. On the other hand, several functions of the visual interfaces can be demonstrated: the *filtering* function, to import only data needed for the analysis, working with data items, that stand for something specific, e.g. numbers which describe months of the year, starting with 0. The user scenario involves also *determinate* and *indeterminate intervals* which are processed with the prototype.

The following user scenario describes the named functions: EasyBiograph is a software used for cooperative acquisition of biography of any person. It allows to visualize important sections and events of a person's life using a timeline [Ilse Arlt Institut für Soziale Inklusionsforschung, 2019]. The persona John, who is a data scientist, would like to import Mr. Hubers biography data into TimeBench and visualize it. The biography data was created in EasyBiograph. The reason for the creation of the TimeBench visualization is that John would like to experiment with the given data and try to get more insights, then the EasyBiograph visualization might provide. Figure 3.1 shows the interface of EasyBiograph and the data of Mr. Huber. The y-axis is showing important dimensions of the person's life: family, living, education, work, healthcare, treatments, others. On the x-axis the user can find the timeline and the age of the person. To save the created data, EasyBiograph uses CSV files, which are shown in the Table 3.1.

event	interval	unknown	determinate	dimension	intervalStartMonth	intervalEndMonth	intervalStartYear	intervalEndYear	description	unknown2	unknown3	unknown4
E	2	16	33	0	2	3	1989	1989	Scheidung de	true	377	137
E	1	18	33	1	0	2	1982	1989	Mietwohnun	false	0	0
E	1	16	33	2	8	7	1989	1990	VS	false	0	0
E	1	16	33	2	8	5	1990	1997	SPZ	false	0	0
E	2	16	33	4	3	4	1990	1990	Blinddarm-Ol	true	399	415
E	2	16	33	4	4	5	1998	1998	Nasenbruch	true	586	398
E	1	18	33	1	8	0	1991	1998	ETW Stiefvat	false	0	0
E	1	17	33	1	2	4	1989	1991	Wohnung Mu	true	353	180
E	1	18	33	1	5	3	1999	2000	LBS	true	599	215
E	1	17	33	1	0	3	1998	2000	Whg. Freund	true	565	180
E	1	18	33	1	4	8	2000	2003	Mietwohnun	true	621	215
E	1	18	33	2	7	0	1997	2000	Lehre	false	0	0
E	2	17	33	1	8	9	2003	2003	Delogierung	true	721	180
E	2	18	33	0	0	1	1997	1997	Konflikte mit	true	560	141
E	1	17	33	0	0	2	1996	2003	Div. kurze Be	false	0	0
E	1	18	33	0	3	9	2003	2005	Beziehung mi	true	720	146
E	2	17	33	0	6	7	2004	2004	Tod d. Tochte	true	736	103
E	2	16	33	4	2	3	2000	2000	Alkoholvergif	true	619	452
E	2	16	33	4	10	11	2000	2000	Alkoholvergif	true	648	428
E	2	17	33	2	0	1	2000	2000	Lehrabbruch	true	633	265
E	1	16	33	5	1	11	2000	2001	AMS	false	0	0
E	1	16	33	3	0	0	2002	2003	Div. Hilfsarbe	true	633	340
E	1	18	33	5	0	0	2002	2005	Sachwaltersc	true	661	515
E	1	17	33	5	0	7	2004	2004	JWF	true	709	480
E	1	17	33	5	0	11	2005	2005	JWF	false	0	0
E	1	16	33	5	0	5	1989	1997	Sprachheilpa	false	0	0

Table 3.1: EasyBiograph export file [Ilse Arlt Institut für Soziale Inklusionsforschung, 2019]

The rows, which start with the "E" entry are describing an interval or an instant date of the event. All other lines that do not start with an "E" can be ignored. Following columns are relevant for John:

- *Column 1*: Only rows with an "E" are signalizing that the row is a time event. All the other rows can be ignored
- *Column 2*: Integer indicating if the time entry is an interval (1) or an instant (2)
- *Column 4*: Integer indicating if the interval is indeterminate (32) or determinate (33)
- *Column 5*: Integer indicating the dimension (0: family, 1: accommodation, 2: education, etc.)
- *Column 6*: Integer indicating the interval start month (0: January, 1: February, 2: March, etc.)
- *Column 7*: Integer indicating the interval end month (0: January, 1: February, 2: March, etc.)
- *Column 8*: Integer indicating the interval start year
- *Column 9*: Integer indicating the interval end year
- *Column 10*: The description of the event

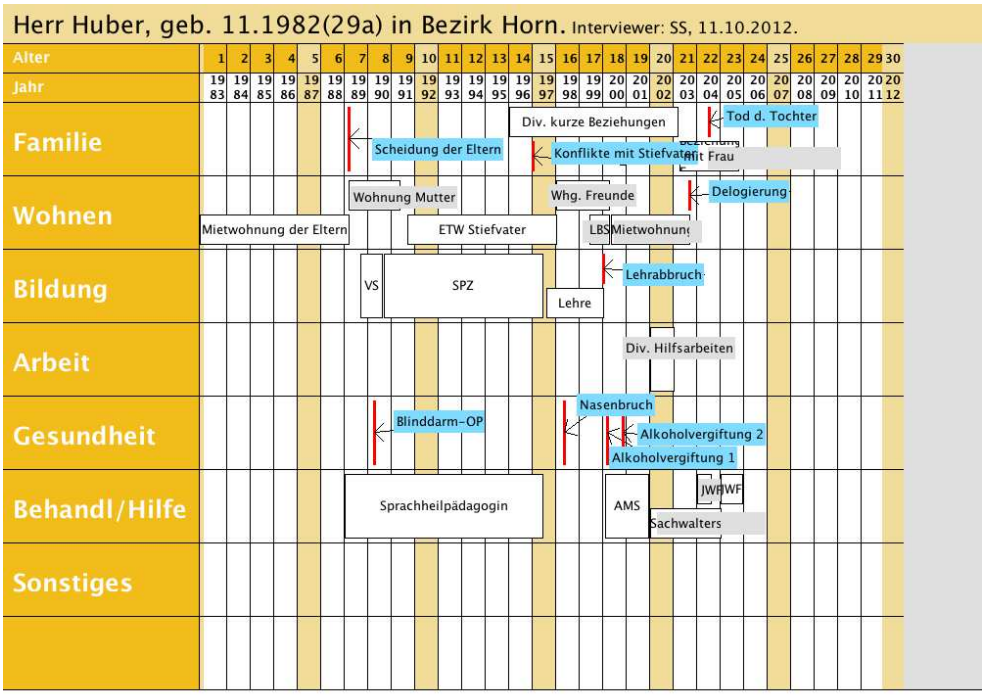


Figure 3.1: EasyBiograph interface [Ilse Arlt Institut für Soziale Inklusionsforschung, 2019]

3.2.2 User Scenario 2 - Sandbox

This user scenario is intended to show an use case in which the user is able to carry out certain arithmetic operations directly in the dataset and can also form different views on the same data (e.g. one timestamp allowing monthly and daily views on the data).

"Sandbox" is a store selling toys for children between 3 and 12 years. The records of the sold toys are stored in the software at the time of the sale. The software has an export function for the accounting data. The recorded data of the sold toys is thus stored in a CSV file, which may be further processed. The owner of the store (Olivia) would like to know on which days of the week the most turnovers are made, and also, which months of the year have the highest turnovers. To visualize the data, she would like to import the data into TimeBench. Table 3.2 shows the export example.

Following columns are relevant for Olivia to import the CSV data correctly into TimeBench:

- *timestamp*: Timestamp in datetime format showing the time of the purchase
- *grossPrice*: Price of the item
- *quantity*: Number of purchased items

id	invoiceNumber	timestamp	store	vendor	item	grossPrice	tax	currency	quantity
156778	156298771	24.01.2019 12:15	VIE18	ASF	P552TNP	62,99	20	E	2
156779	156298772	24.01.2019 13:25	VIE18	ASF	A812TIS	41,99	20	E	3
156780	156298773	24.01.2019 14:35	VIE18	ASF	D721WEY	49,99	20	E	3
156781	156298774	24.01.2019 15:45	VIE18	ASF	T257IXF	71,99	20	E	3
156782	156298775	24.01.2019 16:55	VIE18	ASF	Y105NLN	50,99	20	E	2
156783	156298776	25.01.2019 09:15	VIE18	ASF	Q542LOV	97,99	20	E	3
156784	156298777	25.01.2019 10:22	VIE18	ASF	D625YNT	60,99	20	E	1
156785	156298778	25.01.2019 11:29	VIE18	ASF	A473JNQ	86,99	20	E	1
156786	156298779	25.01.2019 12:36	VIE18	ASF	B511UPL	58,99	20	E	1
156787	156298780	25.01.2019 13:43	VIE18	ASF	C417LCG	95,99	20	E	1
156788	156298781	25.01.2019 14:50	VIE18	ASF	T025CDV	43,99	20	E	2
156789	156298782	25.01.2019 15:57	VIE18	ASF	F071YOB	65,99	20	E	1
156790	156298783	25.01.2019 17:04	VIE18	ASF	J348ZBL	38,99	20	E	1
156791	156298784	25.01.2019 18:11	VIE18	ASF	R758ALQ	56,99	20	E	3
156792	156298785	26.01.2019 09:15	VIE18	ASF	H614UXB	43,99	20	E	3
156793	156298786	26.01.2019 10:00	VIE18	ASF	C978DKY	79,99	20	E	3

Table 3.2: Sandbox export file example

3.2.3 User Scenario 3 - HemingwayFM

The HemingwayFM user scenario shows the interaction between timestamps and spans. The user can form intervals, that consist of a specific time and duration. It also builds on the context of the data. The dataset does not show that every row is committed to a specific date. The user, who has additional knowledge about the available data, is able to add additional data to the dataset and create even more valuable dataset.

The online radio station "HemingwayFM" is famous for the good cultural program and playing classic music in a small city in Illinois (US). Every day on each hour between 6 a.m. and 6 p.m. the radio presenter is telling a short story, which is quite popular among local listeners. Since January 1st, 2017 00:00:00 the radio station has a software (StoryTracker), which makes an entry in the database automatically every hour, saving the hour of the day, the length of the presented short story, the number of audience during the story and some additional information. The radio station owner (Karen) would like to get insights into the data, which gets generated by StoryTracker. She would like to understand, if the length of the story, the time of the day or the time of the year make influence on the number of the audience. To get the named insights, Karen exports the data into a CSV file and wants to visualize the data using the software library TimeBench.

The CSV file (Table 3.3) has following columns, which are relevant for the import into TimeBench:

- *id*: Sequential number providing uniqueness of the data entry
- *hour*: Integer standing for the hour of the day, starting on January 1st, 2017

id	hour	storyLength	storyId	audienceCounter
1	0	0	STO4985	0
2	1	0	STO0068	0
3	2	0	STO1997	0
4	3	0	STO5883	0
5	4	0	STO7070	0
6	5	0	STO9286	0
7	6	321	STO8672	14336
8	7	350	STO4190	14091
9	8	342	STO9284	11963
10	9	219	STO7321	16588
11	10	246	STO4764	18028
12	11	190	STO4076	21427
13	12	238	STO5077	10270
14	13	405	STO3615	14456
15	14	382	STO1499	21412
16	15	393	STO7896	11995
17	16	275	STO5082	11660
18	17	209	STO2873	17823
19	18	259	STO9042	14335
20	19	0	STO8186	0

Table 3.3: HemingwayFM export file example

- *storyLength*: The length of the short story in seconds
- *storyId*: ID representing the story
- *audienceCounter*: The number of listeners during the time interval

3.3 Prototype Requirements Given by User Scenarios

After it became clear, which user scenarios have to be covered, multiple basic questions, regarding the design of the visual assistant, came up. First question: Which functions should the software include, based on user scenarios? This question describes exactly the functionalities which the user scenarios require to fulfil the tasks. The following 2 questions have less functional, but more design in the foreground. By which graphic representation you can visualize arbitrarily nested objects? The answer to this question must satisfy both software architectural and visual requirements. The last question provides information about the user's interaction with the software: By what habitual interactions the user should be able to interact with the objects?

After the user scenarios have been set, it was clear which functions have to be realized in the import assistant software:

- There should be distinction possible between TemporalObjects (object which stores application data with reference to a TemporalElement) and TemporalElements

(interfaces for time primitives - instants, intervals, spans)[Rind, 2017], with a n:1 relationship between those. That means, every TemporalObject has exactly one TemporalElement in TimeBench and the importer should support this architecture

Regarding aspects of time-oriented data, the import assistant framework should support multiple functions:

- The software should support the import of data tables, which consist of columns representing time-oriented data. For example, columns consisting of integers, which stand for years (1999, 2000, 2001, etc.), months (01 stands for January, 02 stands for February, etc.), etc.
- Especially, when there is no context about the time-aspects, the user should be able to enrich the imported data. For example, if the imported data does not have a valid reference to the time components, the user should be able to insert the time reference and enrich the data. In particular, the user can specify that each line of the imported data stands for a week, e.g. beginning with June 1st, 2019
- The prototype has to support different granularities. For example, the user must be able to specify that every row is one decade, year, month, week, day, hour, minute, etc. referenced to a specific start date and time
- With an additional feature, it should also be possible to merge data from multiple columns. That could help the user to create a valid date and time when the date and time information are in different columns (column 1: 01.06.2019, column 2: 23:23:59)
- In addition to instants, it should also be possible to import other time primitives, like intervals (determinate and indeterminate) and spans, into the framework
- Sometimes the data, which has to be imported, by using the prototype, has rows which have to be handled differently. For example, some rows of the data are describing some points in time (instants) and the others are describing intervals (ranges between 2 instants). The prototype should include some kind of filtering function to support the named case
- The user should not only be able to import continuous, but also cyclic data into the software

To assist the user to configure the import of the data, additional requirements on the prototype occurred:

- The user should get feedback and an overview of the progress of his or her configuration. While he or she is configuring the import description, the prototype should give him or her an overview about the already used data (columns) and also show him warnings/errors about the wrong configuration

- The imported data types should be shown to the user in some way
- The repetitive work should be avoided
- The result of the configuration should be shown to the user, who should be able to verify the created import description

3.4 Mockups

The reason for mockups is mostly to specify the user interface features and functions in a simple "quick and dirty" way [Rivero et al., 2010, 2011]. In this chapter a part of the mockups is presented, which describes the design process in the first steps. For each of the scenarios described, the mockups show how the user reaches his or her goals, or better, what steps are required to fulfil the requirements for the given import result.

Mockups were created for each of the three user scenarios. For the first user scenario, every interaction (every click) of the user and the subsequent reaction of the system was displayed. As a result, the mockup process resulted in 69 images for the first user scenario. After the mockups for the first user scenario already showed the most user interfaces, not every interaction of the user had to be displayed for the remaining user scenarios. The representations in the mockups for scenarios 2 and 3 showed only the different configuration settings for the different requirements.

Since the internal representation of the TemporalDataset in TimeBench is done by graphs, we saw as one of the alternatives to represent TemporalObjects and TemporalElements by graphs too. Figure 3.2 shows the mockup of the start screen of the prototype. This screen shows up immediately after the user has started the software and has selected the CSV file to import using the OpenFile menu. If the CSV file is corrupt, the software will issue an error message and it will close automatically. The lower part of the screen shows the imported CSV file in a data table view. The column headers show the names of the columns. The data type is displayed in the form of symbols for each column and is placed over the column header. What is not visible in the mockups is, that the data types are automatically recognized by the software. The bar above the data-type-symbols gives the user the possibility to display the hidden columns and also to show more or less data from the CSV file. These functions were designed to prevent the user's cognitive overload and increase the software's performance. The largest part of the screen represents the place needed for the composition of the TemporalDataset graph. When the program starts, the root node is already added to the TemporalDataset and displayed to the user. The left part of the screen shows the object catalog. In this area graphs can be pinned from the main area for later reuse. This functionality should minimize the repetitive user's work. When clicking the "Done" button, TemporalDataset is generated and is accessible for further visualization frameworks. In addition, the specification of the TermporalDataset is getting created and a GraphML file is generated. This button is only active, if a TemporalDataset can actually be generated and that is only the case, when the user's configuration is error-free.

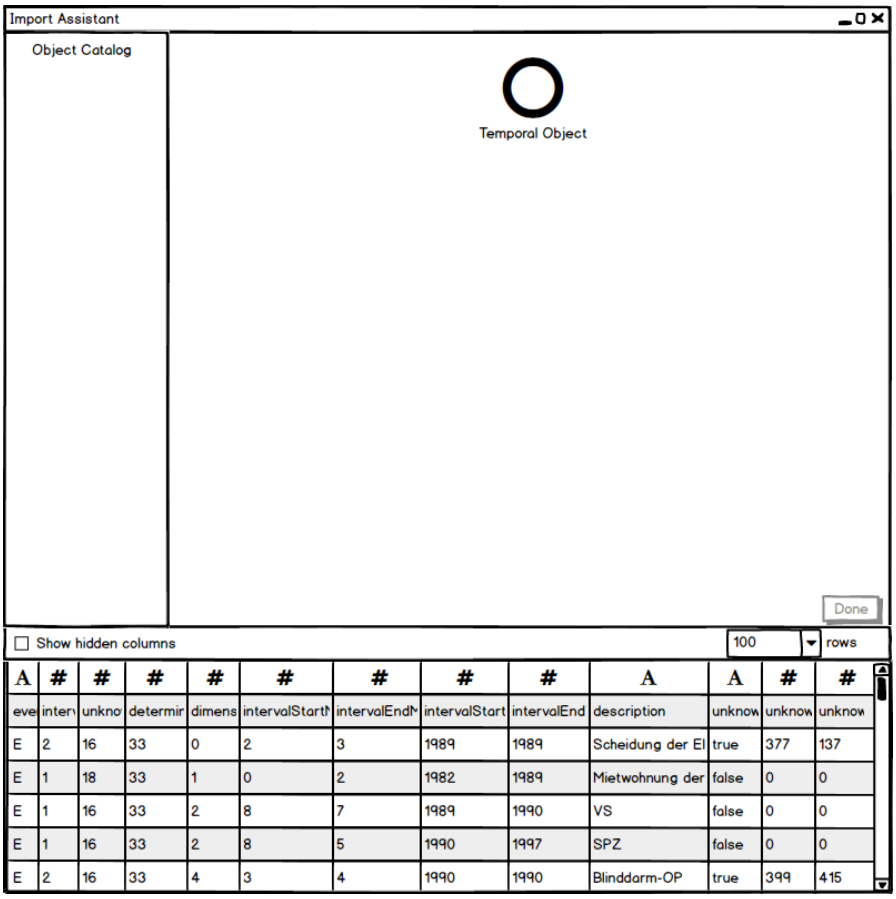


Figure 3.2: Mockup: main screen

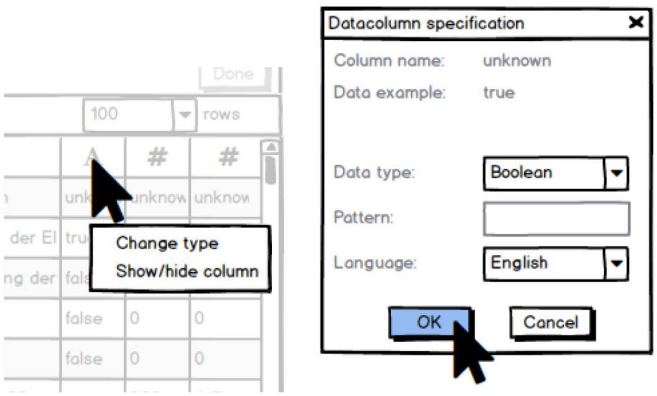


Figure 3.3: Mockup: data table context menu (left) and data type change window (right)

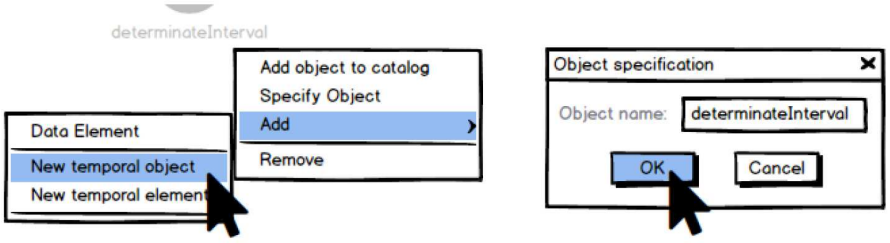


Figure 3.4: Mockup: temporal object context menu (left) change object name window (right)

Figure 3.3 (left) shows the context menu, which appears when the user clicks on the columns of the data table with the right mouse button. The context menu allows the user to show and hide the columns. In addition, the data types of the columns can be adjusted. When clicking on the "Change type" option, the window appears in which the data type can be adjusted (Figure 3.3 (right)). The window shows which column has been selected, including a data example that comes from the first row of the current column. The user has the possibility to choose a new data type from "text", "numeric", "boolean" and "datetime". If the data type "datetime" is selected, the pattern, in which the date is available, can also be described, for example "yyyy-MM-dd HH:mm:ss". The language selection function is limited to the data type boolean, since the value can be given in different languages. The software should then be able to interpret the data correctly, depending on the language, which the raw data contains (For example: "English" should be chosen in case when the data column contains "True" and "False". Otherwise when the data column contains values "Wahr" and "Falsch", "German" should be chosen).

Figure 3.4 (left) shows the context menu, which appears when the user clicks on a TemporalObject with the right mouse button. After that, the user has the possibility to add the object (including its children) to the object catalog and to reuse it at some later point. Also he or she is able to change the object name by clicking on the "Specify object" option. Figure 3.4 (right) is showing the window, which allows the changing of the object's name. This appears only, when the TemporalObject is the root node. In other cases, the window contains a submenu, which can be used to change the TemporalObject to an interval, instant or span. The submenu "Add" allows the user to add a data element. Also, it allows to add a new TemporalObject or TemporalElement (instant, interval or span) as a child. If there are some objects in the object catalog, they are appearing in this list and are ready to get added to the selected TemporalObject as children. To remove the TemporalObject there is also a "Remove" function, which is not only removing the TemporalObject itself, but also its children, after the user has confirmed the warning message.

Figure 3.5 shows the result of changing the TemporalObject node type to "interval". The instant children appear automatically, attached to the interval (parent) node. The red color of the nodes symbolizes that the TemporalElements (in this case - instants) have

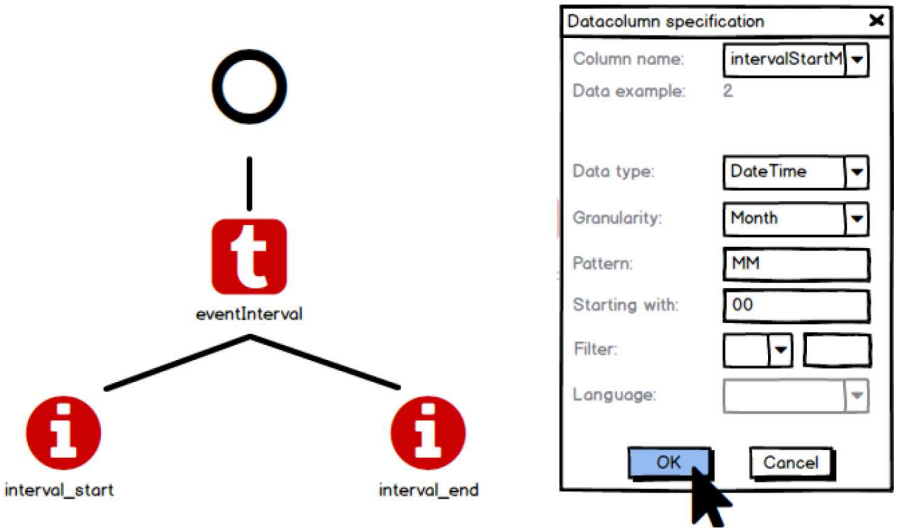


Figure 3.5: Mockup: interval (left) and Datacolumn specification window (right)

not yet been assigned temporal columns from the CSV file. When temporal elements are assigned to temporal columns, not only does the color of the nodes change to green, but also the columns in the data table (Figure 3.2 (bottom)) also turn green. This change of color gives the user an overview of which columns have already been assigned to the nodes and which were not assigned. The temporal columns can be assigned to a node by the user, calling the context menu by clicking with the right mouse button on the nodes and calling the corresponding function. The appearing context menu allows the user to change the node's name, to delete the node and to add a (temporal) data column to the node. By clicking on the "Add data column" option in the context menu, the window for the data column specification shows up (Figure 3.5 (right)). The user is able to choose the temporal data column from the list of all columns. After choosing the right data column, the data example is changed automatically, showing the value of the first data cell of the chosen column. As next step, the user is able to configure the data type and granularity. At the granularity selection, the user has the choice between: "milliseconds", "seconds", "minutes", "hours", "days", "weeks", "months", "years" and "decades". After choosing the appropriate granularity, the user can insert the pattern for the data column values. The values, which are getting imported into the software from the current column looks like following: 00 - January, 01 - February, ..., 11 - December. By using the "Starting with" text field, the user is able to specify that the numeric values, which stand for months, do not start with 01 for January, but with 00. Using the filter function, the user is able to specify the allowed months, which are getting imported. The first drop-down field allows following filtering operators: "<", "<=", "=", "<>", ">", ">=". The text field next to it can be used to enter the operand.

Every instant can contain up to 2 data columns with one data column containing the date and the other the time.

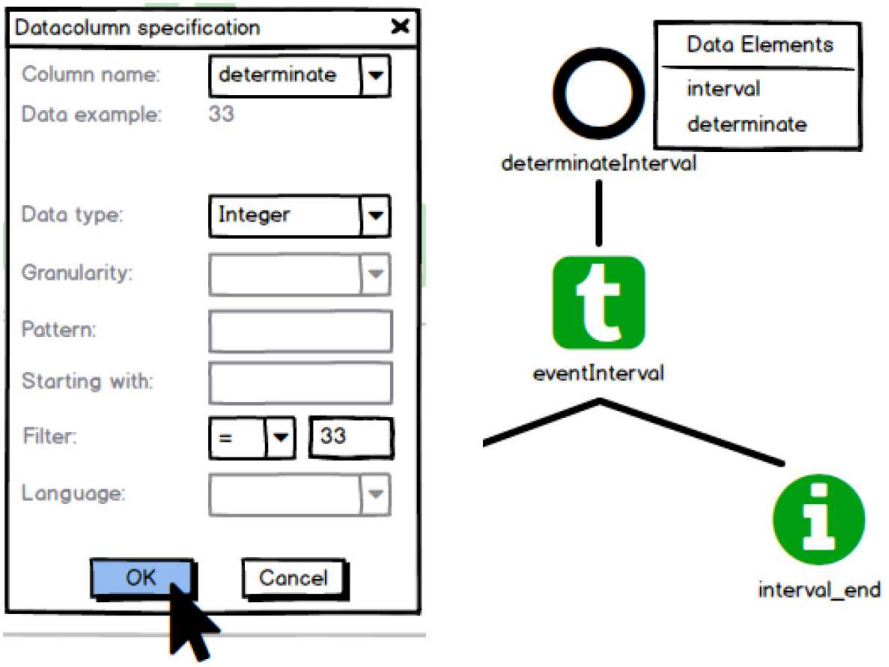
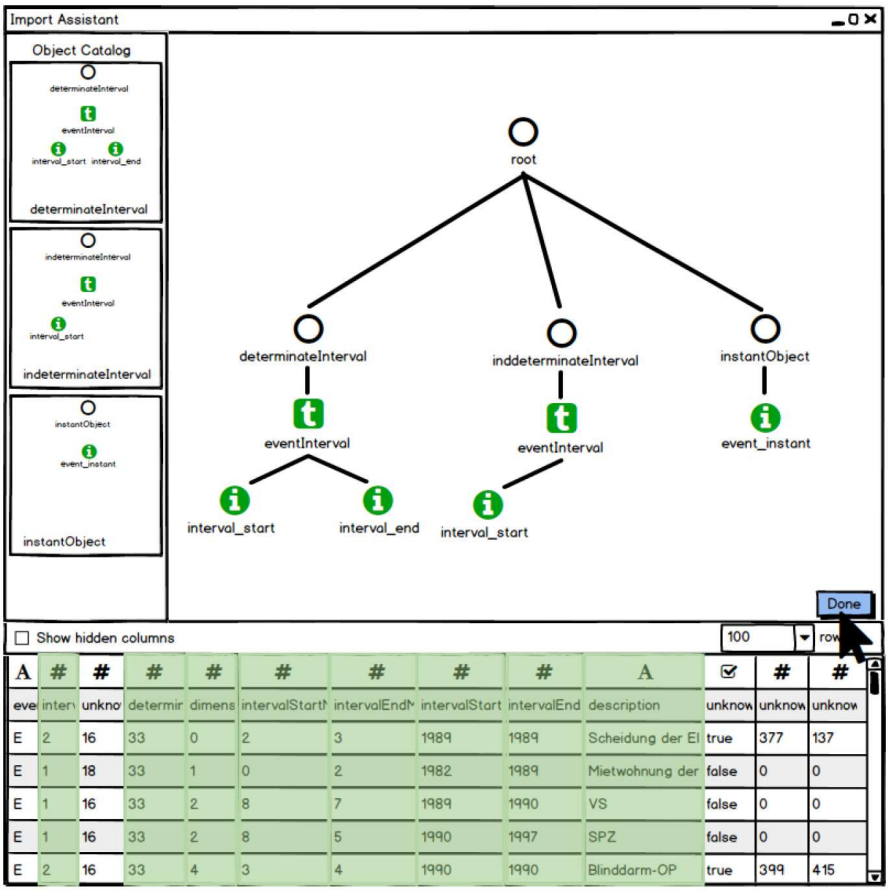


Figure 3.6: Mockup: Data element specification (left) and data element representation (right)

Figure 3.6 (left) shows the data element specification window, which can be called by clicking on the TemporalObject context menu on "Add" > "Data Element" (Figure 3.4 (left)). This window can be used to link the data elements to the temporal data, which has been specified in the children TemporalElements. First, the user can choose the data column to insert. After choosing the column, a data example is shown to the user. The next step, which has to be done by the user, is to select the appropriate data type for the selected column. This selection must be done when using the filter below. The filter function can be used to import only a subset of raw data, which is fulfilling the filter requirements. By choosing the data type "Integer", the user is able to choose following filter operators: "<", "<=", "=", "<>", ">", ">=". If the chosen data column contains only textual values, following operators are available: "=" and "<>". In the shown figure, the user is setting the filter for the column "determinate", importing only those rows, which contain the value "33" in the column "determinate". The reason for the users action is the fact that the user has the knowledge, that the value "33" of the column "determinate" marks the rows as a determinate interval data in the raw data. It is possible to add multiple data elements to the TemporalObject having their own filters. Figure 3.6 (right) shows the representation of the data elements, which have been added to the TemporalObject. There is also a function, which can be used to delete the added data elements. This function can be called from the TemporalObject context menu and is appearing only if the TemporalObject contains more then 0 data elements.



A	#	#	#	#	#	#	#	#	A		#	#
eve	inter	unkno	determir	dimens	intervalStartN	intervalEndN	intervalStart	intervalEnd	description	unknow	unknow	unknow
E	2	16	33	0	2	3	1989	1989	Scheidung der El	true	377	137
E	1	18	33	1	0	2	1982	1989	Mietwohnung der	false	0	0
E	1	16	33	2	8	7	1989	1990	VS	false	0	0
E	1	16	33	2	8	5	1990	1997	SPZ	false	0	0
E	2	16	33	4	3	4	1990	1990	Blinddarm-OP	true	399	415

Figure 3.7: Mockup: finished import configuration

Figure 3.7 shows the mockup of the finished import configuration for the first user scenario (Chapter 3.2.1). The user has created 3 TemporalObjects. The first was created to import only determinate intervals (intervals, which have a start and an end instant). The filter, that has been used was applied on data column "determinate" on value "33". The second TemporalObject has been created for indeterminate intervals. In this case, the same filter has been used, but this time on the data column "determinate" was filtered on the value "32". The named value stands for indeterminate interval. The third TemporalObject was inserted to import all events, which are no intervals. The user has set an filter on the second column ("interval"). Every row, which has the value "2" in the column "interval" represents an event, which is not an interval. The object catalog displays the objects "cached" by the user. Since the first and second TemporalObject differ only in the filter, the user can benefit from the catalog function. The procedure to perform the entire configuration for the first user scenario would be the following:

- create determinateInterval TemporalObject,

- pin the created object to the object catalog,
- add the object from the catalog to the work space again,
- change the filter option to indetetminate interval,
- rename the object to "indeterminateInterval",
- pin the new object to the catalog,
- create the "instantObject" TemoralObject,
- pin the object to the catalog,
- create new root object and add all object from the catalog as children

CHAPTER 4

Implementation

The design of the interactive visual interface took place in several steps. First, the basic requirements for the prototype have been set. After the basic requirements were clear, we created 3 user scenarios to describe the scope of the work and to determine the remaining requirements. The next step was to create mockups, in order to show the way how the user interacts with the software, and also, to specify prototypes features and functions. After the mockups were completed, there was already a clear idea of what the prototype will look like, what features it will have, and how the user is going to interact with the prototype.

This chapter describes the next step, the process of implementation. The prototype was implemented in Java 8. As an integrated development environment (IDE), Eclipse version 2018-09 (4.9.0) was used. In order not to have dependencies on various libraries, the entire user interface was created using JavaFX standard objects. To use the TimeBench data structure, the project requires following dependencies: *commons-lang3*, *ical4j*, *log4j*, *ieg-prefuse*, *ieg-util*, *TimeBench*.

Figure 4.1 shows the prototype's class diagram. In following, the important classes will be described and in addition, the revision of the design will be presented.

The entry point of the software is the *Main.java* class. This class offers only a few functionalities. The first step is to ask the user via FileChooser to select the CSV file to import. After the CSV file has been imported, it will be passed to the *MainController.java*. The only function that the *Main.java* class offers, is the getTemporalDataset() function, which retrieves the result of the prototype as a TemporalDataset. Listing 4.1 shows the Java code used to retrieve the TemporalDataset. The function uses the *MainController.java* to retrieve the result. The reason why this function was placed in the code at this point is that the users, who use the prototype as a library, can easily find it. The same function can also be called in in *MainController.java*. The input, that the user has to make when

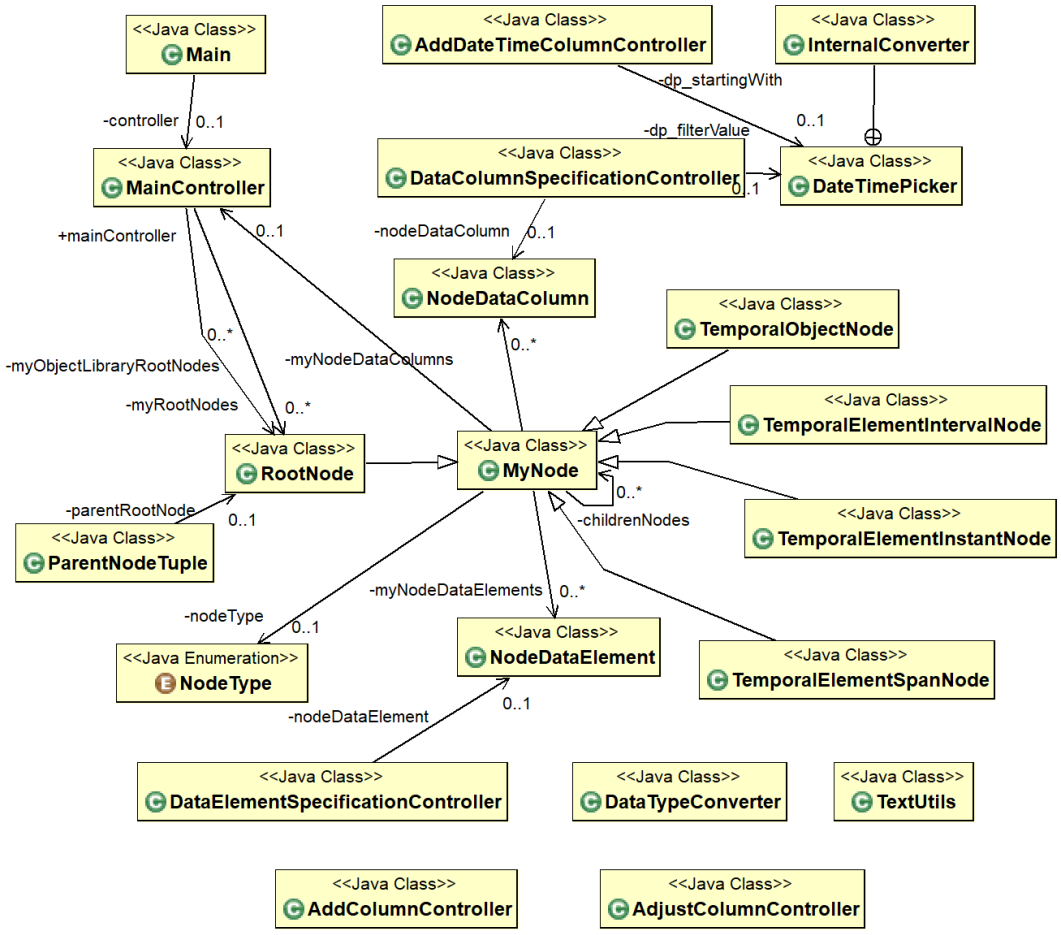


Figure 4.1: Class diagram

calling the library is a CSV file. The output is a TemporalDataset object, which can request via the function getTemporalDataset() in *Main.java* class.

The general data structure and the program flow looks as follows:

- The entry point of the software is the *Main.java* class. This class starts the file chooser, asking the user to select the CSV file, containing the time-oriented data.
- After checking the inserted file, the *MainController.java* class gets called and the CSV file is passed to the class.
- The *MainController.java* class creates the data table, filling it with the data from the CSV file. The wrangling functions can be done directly in the created data table. The main panel of the MainController allows the user create instances of the

MyNode.java class, which represents the different node types (RootNode, TemporalObjectNode, TemporalElementIntervalNode, TemporalElementInstantNode and TemporalElementSpanNode).

- By clicking on the "Done" button, a new TemporalDataset object is created. The nodes on the MainController panel are linked to the wrangled data from the data table and are added to the TemporalDataset.

- Using the getTemporalDataset() function, the user is able to retrieve the TemporalDataset object and use it in the visualization.

```
public TemporalDataset getTemporalDataset()
{
  return controller.getTemporalDataset();
}
```

Listing 4.1: Retrieving of TemporalDataset in Main.java

```
public List<MyNode> getAllChildren(MyNode nod) {
  List<MyNode> allChildren = new ArrayList<MyNode>();

  for (MyNode childNode : nod.getChildrenNodes()) {
    allChildren.addAll(getAllChildren(childNode));
  }
  allChildren.add(nod);
  return allChildren;
}
```

Listing 4.2: Get all children from MyNode in MainController.java

The *MainController.java* class builds the core component of the software. This class is not just the controller for the main user interface, but also forms the basis for the creation and management of the TemporalDataset. The recursive approach allows traversal of the graph, which represents the TemporalDataset. Listing 4.2 shows an example for retrieving all children of a node. By clicking on the "Done" button in the user interface, the function *createTemporalDataset()* is called. For each data table row (in TimeBench context: Tuple) the TemporalDataset graph is traversed and depending on configuration, the data is added to the TemporalDataset, or is ignored because of the active filters. The wrangling of the data table also takes place in this class. Following functions can be found in the *MainController.java* that can be used to adjust the raw data in the data table (*dt_myDT*):

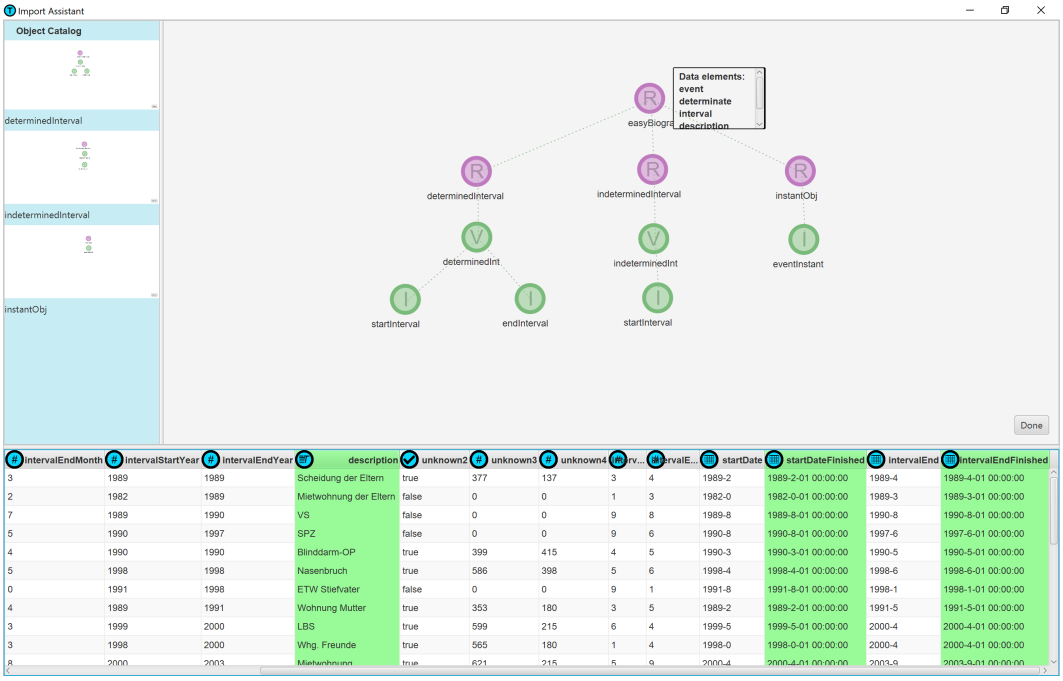
- *adjustColumn(...)*: this function allows the user to increase/decrease numerical values from selected data table columns. Also, it is possible to insert prefixes and/or suffixes to the raw data. This can be useful to create the valid datetime data type (format yyyy-MM-dd HH:mm:ss) from incomplete values. For example, when the data column includes values in the following format: "2019-05", the user is able to create complete dates by adding suffixes like "-01 00:00:00" to create correct datetime "2019-05-01 00:00:00". Following, the created column can be used as a source data column for a TemporalElement object.

- *addDateTimeColumn(...)*: using this function, the creation of additional datetime data columns can be performed. The user has the possibility to decide the start datetime of the first data column value and the steps between the rows (granularity). Startdate "2019-01-01 12:00:00" and granularity "hour" would result in column values: 1st row - "2019-01-01 12:00:00", 2nd row - "2019-01-01 13:00:00", 3rd row - "2019-01-01 14:00:00", etc.

- *createNewColumn(...)*: allows merging from 2 columns into a new one. Not only that date and time columns can be merged to one valid datetime column, also basic arithmetic operations can be applied between 2 numeric columns. This can be useful when the data includes data, which must be calculated before visualization.

Besides *MainController.java* builds *MyNode.java* class the most important class in the software. This class includes several important functions. An important task that this class does, is to represent the TemporalObject and TemporalElement nodes in the user interface with all the functionalities (add, delete, etc.). *MyNode* also supports a parent-child relationship between instances to support the graph representation. Depending on the type of node (enumeration of *NodeType*), a node may contain several instances of the class *NodeDataColumn.java* or *NodeDataElement.java*. Other than *NodeDataElement.java*, which connects non-temporal data columns of the data table to TemporalObjects, *NodeDataColumn.java* class includes the description of the temporal columns attached to a TemporalElement. To support the different functionalities of the nodes, depending of their type, classes *RootNode*, *TemporalObjectNode*, *TemporalElementIntervalNode*, *TemporalElementInstantNode* and *TemporalElementSpanNode* are extending the *MyNode.java* class. The named extended classes include their own context menu and the appearance attributes for the user interface. Since all of the node types share the same functionalities (parent-child relationship, add, delete, translate, etc.), they inherit those from the super class *MyNode.java*. the only reason for the distinction between the RootNode and TemporalObject is the fact that it should be clear to the user that at least one TemporalObject must be present in the TemporalDataset object. The functionalities, behind both objects are the same.

In order to better support the user doing his or her task, Cascading Style Sheets (CSS) files *application.css* were generated that, for example, place correct data type icons in the data table column headers or change the background colors of the already assigned columns.



intervalEndMonth	intervalStartYear	intervalEndYear	description	unknown2	unknown3	unknown4	rv...	ervalE...	startDate	startDateFinished	intervalEnd	intervalEndFinished
3	1989	1989	Scheidung der Eltern	true	377	137	3	4	1989-2	1989-2-01 00:00:00	1989-4	1989-4-01 00:00:00
2	1982	1989	Mietwohnung der Eltern	false	0	0	1	3	1982-0	1982-0-01 00:00:00	1989-3	1989-3-01 00:00:00
7	1989	1990	VS	false	0	0	9	8	1989-8	1989-8-01 00:00:00	1990-8	1990-8-01 00:00:00
5	1990	1997	SPZ	false	0	0	9	6	1990-8	1990-8-01 00:00:00	1997-6	1997-6-01 00:00:00
4	1990	1990	Blinddarm-OP	true	399	415	4	5	1990-3	1990-3-01 00:00:00	1990-5	1990-5-01 00:00:00
5	1998	1998	Nasenbruch	true	586	398	5	6	1998-4	1998-4-01 00:00:00	1998-6	1998-6-01 00:00:00
0	1991	1998	ETW Stiefvater	false	0	0	9	1	1991-8	1991-8-01 00:00:00	1998-1	1998-1-01 00:00:00
4	1989	1991	Wohnung Mutter	true	353	180	3	5	1989-2	1989-2-01 00:00:00	1991-5	1991-5-01 00:00:00
3	1999	2000	LBS	true	599	215	6	4	1999-5	1999-5-01 00:00:00	2000-4	2000-4-01 00:00:00
3	1998	2000	Whg. Freunde	true	565	180	1	4	1998-0	1998-0-01 00:00:00	2000-4	2000-4-01 00:00:00
8	2000	2003	Mietwohnung	true	621	215	5	9	2000-4	2000-4-01 00:00:00	2003-9	2003-9-01 00:00:00

Figure 4.2: The main screen of the prototype

Figure 4.2 shows the main screen of the prototype. In this case, the main screen is showing the solution for the first user scenario (Chapter 3.2.1). The different node types (TemporalObject, TemporalElement - Instant, TemporalElement - Interval) do have different colors and labels inside the objects (R - RootNode, V - TemporalElementIntervalNode, I - TemporalElementInstantNode, O - TemporalObjectNode). Similar to the mockups in Chapter 3.4, the object catalog is placed on the left edge of the screen. Every object that has been pinned to the object catalog has a name, which is placed under the object in the catalog. The green columns in the data table symbolize that the column has already been assigned in the graph. The blue icons in the data column headers indicate the data type of the values in the column.

Figure 4.3 (left) shows the window for adding the data column to a TemporalElement. All the required fields are marked with an asterisk (*). The field "ColumnName" requires the selection of the data column from type datetime. After selecting the column, a data example is shown automatically to the user. The data type is always "Date" and the required pattern is "yyyy-MM-dd HH:mm:ss". These two fields can not be changed, since the system needs the data in exactly that format. In the next step, the user must choose a granularity. In this case, the user has the choice between: "milliseconds", "seconds", "minutes", "hours", "days", "weeks", "months", "years" and "decades". Optionally, also a filter can be set. For example, it can be adjusted, that only the data before June 1st, 2019 should be linked to the TemporalElement. For the selection of time, the JavaFX

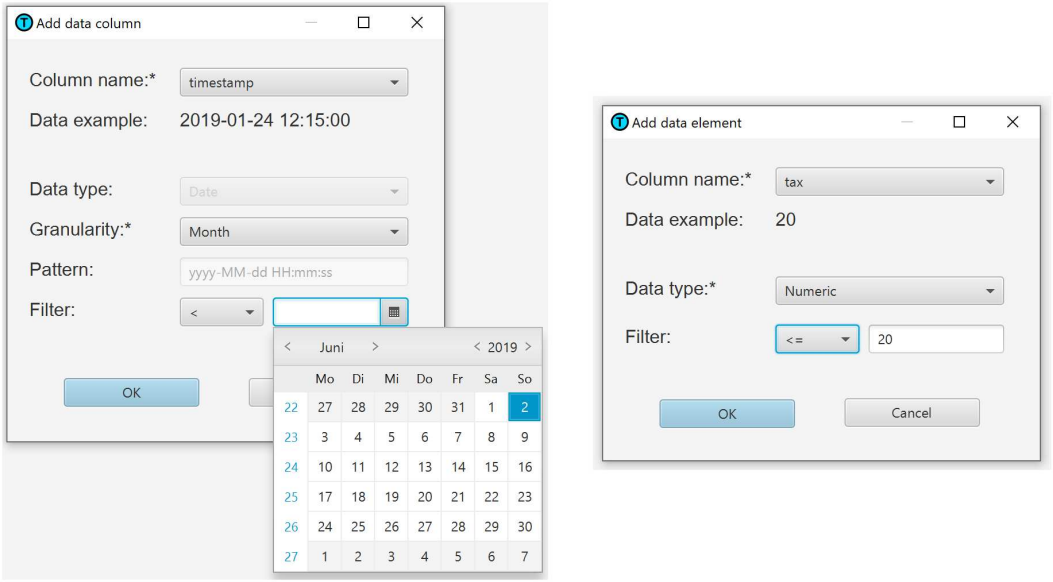


Figure 4.3: Add data column (left) and add data element (right)

DatePicker [1] is used. After applying the changes by clicking on the button "OK", the system verifies if the selected data column values can be converted to datetime and if the pattern is correct. If the verification is positive, the data column is linked to the TemporalElement, otherwise the window remains open and the user is alerted to the errors that have occurred.

The window in Figure 4.3 (right) shows the "Add data element" dialog. Using this window, the user is able to apply non-temporal data columns to the TemporalObject. From the drop down menü "Column name" the user has to select the column, that he or she would like to add. After the selection, a data example, from the first row of the selected column, is shown to the user. Another required selection, that the user must configure is the data type field. Here he or she can choose between "text", "numeric", "boolean" and "datetime". Similar to the "Add data column" window, a filter can be added to the data element. For the numeric and datetime values following operators can be used: "<", "<=", "=", "<>", ">", ">=". For the textual and boolean values only "=", "<>" can be used. This filter helps to add only those tuples to the TemporalDataset, which are fulfilling the requirements, set by this filter.

Figures 4.4 and 4.5 are showing the windows, which can be called by the data table context menu. The reason for their usage is to insert the user's knowledge into the raw data by reshaping the raw data, merging columns, creating completely new columns and therefore creating more valuable data. The window, which is displayed in Figure 4.4 (left) shows a function, which can be used to create a new datetime column. By

[1] https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/DatePicker.html

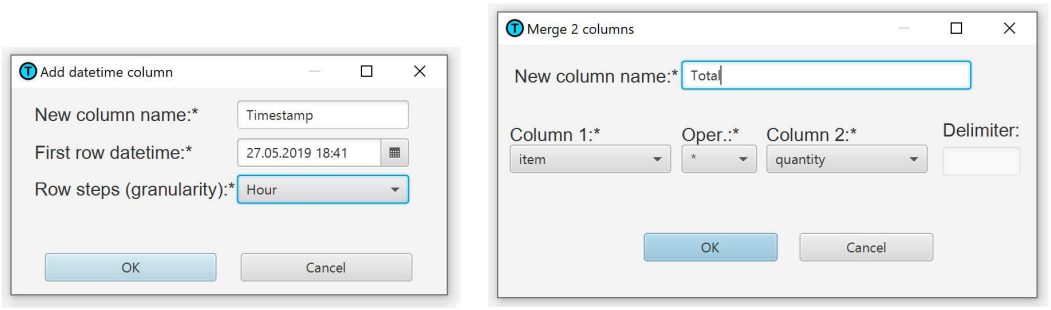


Figure 4.4: Add new datetime column (left) and merge columns (right)

inserting a new column name, the start row datetime and the steps, the user creates a new column, which has the start datetime in the first cell. For all the other values in the column, the function creates continual datetime values, that increase by the value chosen in the "Row steps" field. For example shown in Figure 4.4 (left), the values for the column "Timestamp" would look like following:

- *cell 1*: "2019-05-27 18:41:00"
- *cell 2*: "2019-05-27 19:41:00"
- *cell 3*: "2019-05-27 20:41:00"

Figure 4.4 (right) shows the window used to merge two columns. It can be used to create a new column by concatenating two columns (column "Month" with value "03" and column "Year" with value "2019" would result in "03-2019" by merging the two columns, using the "&" delimiter). Another approach is shown in the Figure. It is possible to merge columns using basic arithmetic functions. However, this function can also be executed on numeric data column values. In the shown case, the user is creating a new column "total" (total price) by multiplying the price per unit with the number of purchased units.

The shown window in Figure 4.5 is representing the "Derive column" function. Using this window, the user is able to create a new column, using the values from the selected column. "Prefix" and "Suffix" functions are adding a static value before or after the cell value. The numeric column values can also be increased or decreased. The shown picture is presenting a case where the user creates a valid datetime column (which can be used as data column for the TemporalElement) by adding a suffix to the cell values. The result for the first row of the new column "intervalStart" will be: "1989-2-01 00:00:00".

4.1 Revision of the Design

During the implementation process a few problems appeared that had to be solved, in order to provide an unambiguous and well defined user experience. It turns out, that

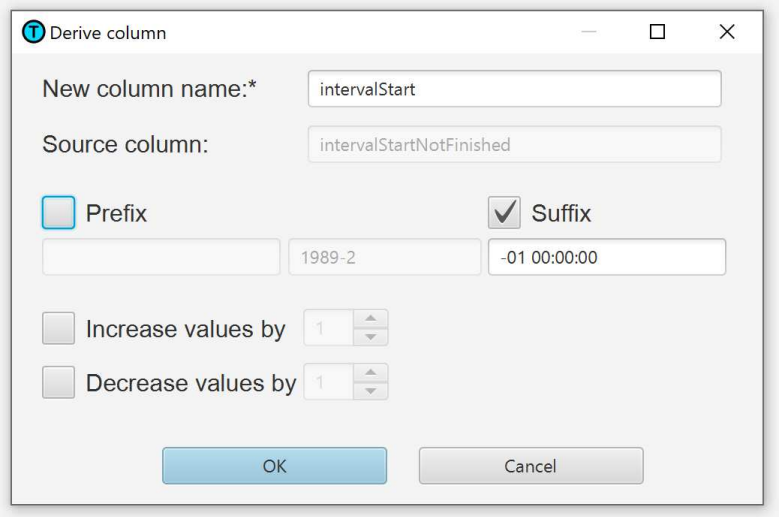


Figure 4.5: Derive column window

although the mockups can be fully implemented, some cases have been overlooked in the mockups. The mockups focus on the perfect performance while creating a configuration for the user scenarios. No consideration was given to cases, when the user, for example, accidentally deleted something, or made incorrect settings.

One of the things that needed to be done better, is the function the user could use to change data types (Figure 3.3 (right)). It was not clear for the user, why this had to be changed. Since this function is only needed for the correct execution of the filters, it has been relocated into the windows, where the filter function is configured and defined as required. This gives the possibility to filter the data of a certain column in several ways. For example, numeric values can be filtered to a specific value, even if they are simultaneously acting elsewhere as a part of a date.

The next window that needed to be adapted, was the "Datacolumn specification" window (Figure 3.5 (right)). For the user it was not clear, which patterns he or she is able to use. It was not clear if the pattern "MM" does mean the granularity "month" or only the month of the date will be "cut out" of the complete date. Also, the field "Starting with" was confusing. The user did not know, if the data would start with this start date, or if the data should be filtered on the set date. Apart from that, specifying multiple date columns for a node was not clear enough. The user could put together a date in the node, which consists of 2 different columns (one is the date and the other is the time), regardless of whether the date is including only year, or year and month. In order to solve that, a different approach had to be made. The user should bring the data into a specific shape before adding the finished configured column to a TemporalElement. In order to do that, several functions have been created that support data wrangling. The user is now not only able to create new columns, but can also merge and modify columns without damaging the original data (Figures 4.4 and 4.5). This approach gives

much more overview of the imported data. In addition, the user is able to maintain the overview of linked data easier. Figure 4.3 (left) shows the improved window. The user only has to select the datetime column and the granularity with which the data is read out. The data type expected from the data column is a datetime with the special pattern ("yyyy-MM-dd HH:mm:ss").

CHAPTER 5

Evaluation

After the implementation was done, the next step was the evaluation of the created software. The evaluation should identify any software errors that may occur, as well as evaluate the user experience and usability of the prototype. In addition to these results, the new requirements which appeared during the evaluation, that, if they are not implemented immediately for various reasons (scope of the work, minor importance for the usability, etc.), are stored as ideas for future work. The evaluation was carried out in several steps. As the first type of evaluation, attention was paid to avoiding programming errors during the implementation. Each function was individually tested during the implementation. After several coherent functions were implemented, they were again tested in combination. After the software was completely implemented, it was fully tested, paying special attention to the correct execution of the user scenarios. After the evaluation was finished by the developers, usability inspections were conducted by 3 usability experts. The next chapter describes the process of usability inspections, which have been performed.

5.1 Usability Inspections

In order to improve the software usability, Nielsen has provided a list of heuristics, based on a factor analysis of the explanations, which best explain usability problems[Nielsen, 1994a]. They are:

- visibility of system status
- match between system and the real world
- user control and freedom
- consistency and standards

- error prevention
- recognition rather than recall
- flexibility and efficiency of use
- aesthetic and minimalist design
- well-structured features that are easy to discriminate
- use of default values

Building on these heuristics, usability inspections were conducted with usability experts. An usability inspection generally proceeded as followed. The first version of the software was reviewed successively by 2 experts. The final version was then reviewed by an expert to determine, if the issues, that appeared during the first usability inspections, were resolved and if usability was improved. For this, the experts were invited to individual sessions. First, the purpose of the meeting was explained to the person, which was the usability inspection, done by usability experts, for the formative evaluation. To record usability inspections and analyze them later, the desktop was recorded, using the software VLC media player [1]. In addition to the video recording, audio was recorded using smartphone microphone and notes were taken. Because the session focus was not the learnability of the software usage, the software was explained in detail and afterwards functionalities were presented. To summarize the presentation, a description sheet about the project has been compiled and given to the expert as a cheat sheet (Figure A.1). In order to explain how to use the software, the facilitator performed one user scenario in front of the experts. After the developers had performed the first user scenario using the prototype and the experts' questions were answered, the usability experts were tasked with implementing the remaining 2 user scenarios themselves. While the tasks were being performed, attention was paid to how the experts use the software to achieve their goals. In addition, the comments of the experts were noted, since they were very important for the analysis and for the improvement of the prototype. After the tasks were performed correctly by the experts, a final interview was conducted in which the main points were discussed again, with a strong reference to the heuristics of Nielsen. Finally, the persons evaluated the software according to the heuristics. The duration of the sessions was between 45 and 60 minutes.

5.2 Usability Inspection Results

After the first two sessions, multiple improvements have been noted by the experts, regarding the design, usability, data wrangling and results. The complete list of the improvements can be found in Appendix B.

[1] https://www.videolan.org/vlc/index.de.html

We sorted the suggestions into categories and rated them according to severity and implementation effort. The following categories are derived from the suggestions:

- *Design*: Those suggestions fall into this category, whose focus is on design. For example the design of the icons and colors
- *Usability*: Suggestions concerning the usability of the prototype. An example for the usability issue is the missing of the default values for the dialog masks
- *Data wrangling*: Describes all issues that occur during the conversion of the raw data.
- *Result*: All problems regarding software output. For example, incorrect export of the data.

After the first two sessions, we had a list over 55 suggestions of improvement. Regarding the design of the prototype, here are some of the suggested improvements: "adjustment of the icon contrast", "too small table preview" and "adding toolbar to the table preview, containing the options from the context menus".

Regarding the usability of the prototype, some of important suggestions for improvement were: "to disable the "Done" button when the TemporalDataset can not be created, because of the wrong configuration", "the change of the data columns and data elements should be enabled, not only the deletion of all", "default values should be added to prevent errors", "missing data provenance" and "missing function for saving sessions".

The category "data wrangling" had beside improvements also a few software bugs that had to be repaired: "it is possible to merge columns with an empty operation, which leads to an error" and "it is possible to create new data table columns with a new of already existing data column"

Regarding the result category, there were two issues. Wrong edges were put on the nodes and the names of the instants were not shown in the output and because of that, they could be named like the data column they are linked to.

There were 2 criteria for choosing which improvements should be implemented in scope of this work, which should be done in the future and which do not have relevance to this work. The first criterion was whether the improvement fits into the scope of the work, or not. An example of an improvement, that was not further developed for this reason, was the ability to save sessions to be able to carry on with the work at a later point. Even if this requirement is very useful, it was not intended in the first place and in addition, the effort required to ensure this functionality is not proportionate to the added value. Additionally, the same applies to the undo/redo functionalities. However, these functionalities should be implemented in the future works.

One of the criteria for selecting the tasks to be implemented was the comparison of the benefits of the functionality and the effort involved in the implementation. If the

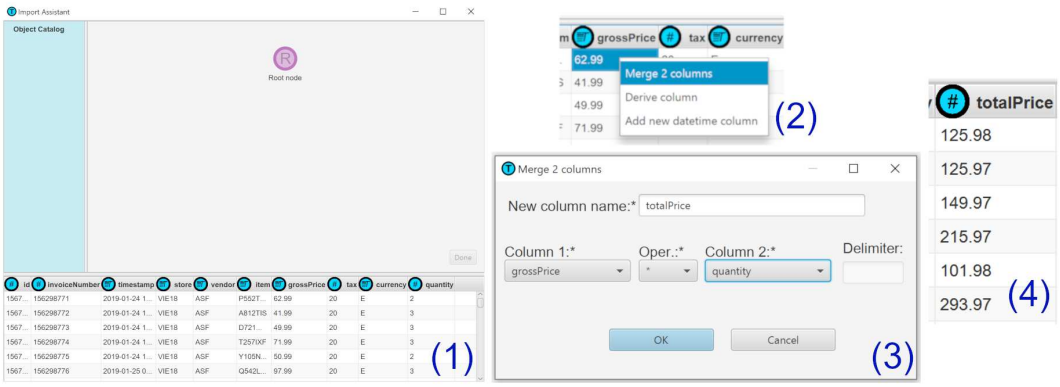


Figure 5.1: The walkthrough of the second scenario - new column

effort was much greater than the actual benefit of the functionality, the task was not implemented. One of the issues was, that the catalog objects are small and hard to read. The function for the display of the catalog objects was implemented by creating a screenshot of the main panel and storing it in the object catalog as a picture. Because for the most tasks, a few number of nodes is needed, there is a lot of white space in the screenshot. The solution would be, to search for all sub objects in the panel and to save the positions of the sub objects. Then derive the maximum positions in all directions and create a screenshot only with these positions (cropped image). Because the name of the root takes place under each object in the object catalog, the effort for this function is greater than the benefit.

All the improvement proposals, which were not excluded by the named criteria, were implemented. That were all functionalities that bring much to the usability and are in good proportion to the effort, and also, all programming errors that were noticed during testing.

After the implementation, the third usability inspection took place. In this session, the reworked prototype was used. This iterative approach has helped the prototype reach a status, where it can be used productively, even if there are still some functionalities that can be made even better in the future.

5.3 The Walkthrough

After the usability inspections have been performed, the next step was to evaluate the utility. More accurate, the fulfilment of the user scenarios should be evaluated by performing a walkthrough. This chapter shows how to perform the second user scenario "Sandbox" (described in Chapter 3.2.2) using the interactive user interface.

Figure 5.1 (1) shows the main screen the user gets, when he starts the software and imports the CSV file. In the lower part of the main window, he can see and scroll through the imported CSV data in the data table. After the user has not yet stored any

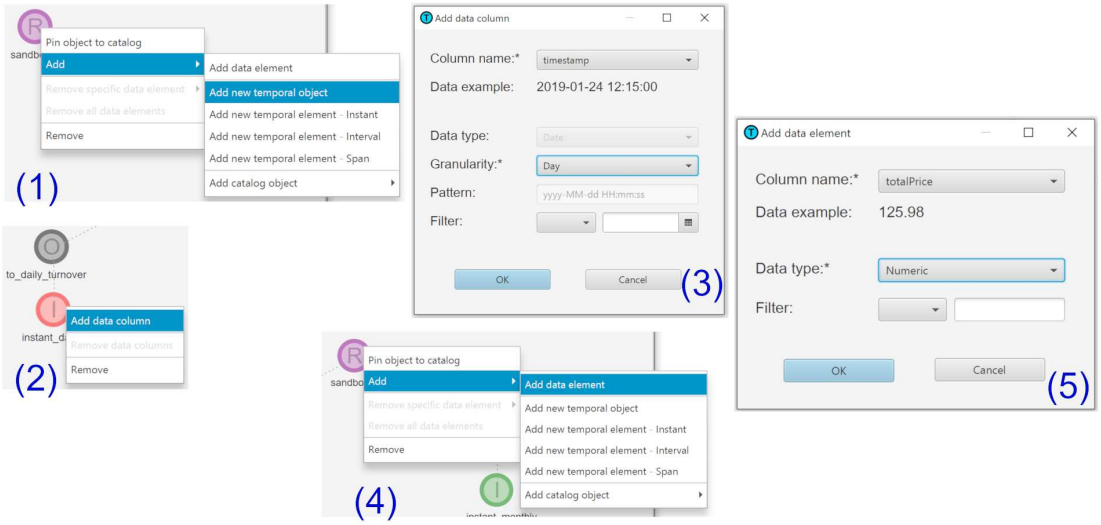


Figure 5.2: The walkthrough of the second scenario - adding nodes

objects in the object catalog, no objects are displayed in this area. The main panel shows already one root node, which is created by default on the program start. The imported accounting data stores the time of the purchase, the price of the bought items and the quantity of the purchased items. The first thing the user wants to do is to generate the total price from each line of the accounting data. To make this possible, the user has to create a new column, which consists of 2 different columns "grossPrice" and "quantity". Figure 5.1 (2) shows the context menu the user gets, shown by clicking with the right mouse button on any of the data columns of the data table (shown in Figure 5.1 (1) - bottom). By clicking on "Merge 2 columns", the form from Figure 5.1 (3) appears. The new column, which the user wants to create should be a product of the gross price and the quantity. He or she gives a name for the new column (in this case "totalPrice"). The first column, that has to be multiplied is "grossPrice" and the second column, that the user selects from the drop down menu, is named "quantity". The operator multiplication "*" describes the operation, which the user wants to execute on the data to create the new column. By clicking on the "OK" button a new column appears (Figure 5.1 (4)) in the data table, showing the total price as a product of the gross price and quantity.

The next step, that the user has to perform is to create the 2 temporal objects and for each temporal object one temporal element (instant). The reason why he or she has to create 2 objects is, because the business owner Olivia wants to know on which days (1) and in which months (2) of the year the most turnover is made. To do that, the user creates a new TemporalObject by clicking on the root node with the right mouse button and choosing the "Add new temporal object" option (Figure 5.2 (1)). As a result

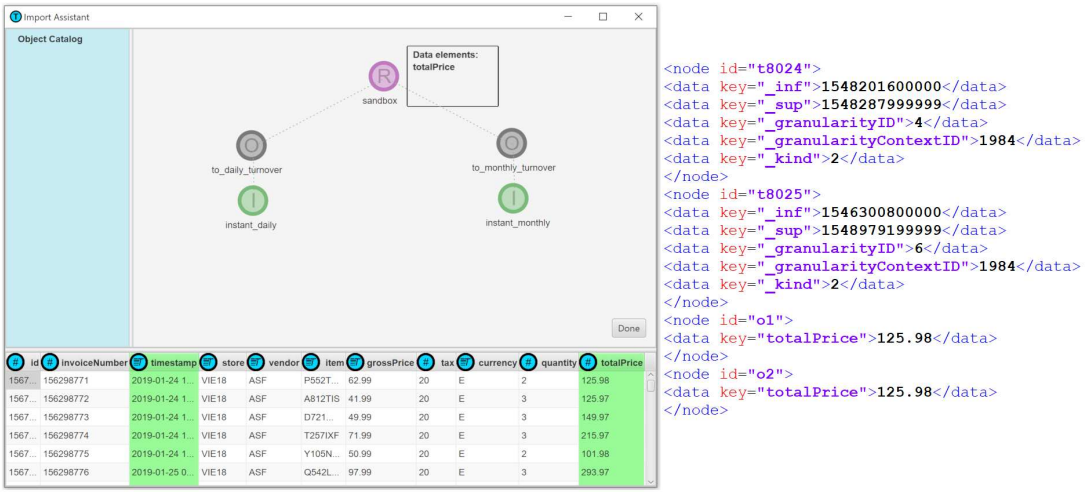


Figure 5.3: The walkthrough of the second scenario - the result

of this action, a new TemporalObject node is created on the main panel. Using the drag-and-drop function, the user is able to move the node on the panel. By clicking into the label below the node, the user is able to change the object's name. Applying the same "add object" action on the new TemporalObject, the user is able to create a new TemporalElement - Instant, as a child object of the TemporalObject node. To link the data to the instant, the user clicks with the right mouse button on the instant node and chooses "Add data column" option (Figure 5.2 (2)). The form shown in Figure 5.2 (3) appears. The user chooses the "timestamp" column from the drop down menu and the data example "2019-01-24 12:15:00" is shown to the user. For the first instant, the user chooses the granularity "Day", not adding a filter to the data. Since Olivia wants to find the months too, in which the highest turnover is made, she creates a new TemporalObject and TemporalElement (instant) as a child. This has to be done in the same way as described above. The only difference this time, is that the user has to choose "Month" as the granularity in the "Add data column" form (Figure 5.2(3)). The last thing, that has to be done, is the link between the created objects and the total price data. Figure 5.2 (4) shows the context menu option "Add data element", which can be used to link the non-temporal data to the temporal objects. After choosing the "Add temporal element" option in the context menu of the root TemporalObject, the form, shown in Figure 5.2 (5), appears. Olivia chooses the "totalPrice" column from the drop down menu and the data type "Numeric" to be added correctly to the TemporalDataset.

Figure 5.3 (left) shows the result of the import configuration. It results in one root object, 2 TemporalObjects and 2 TemporalElements (instants). The instants are linked to the

same column from the CSV file (timestamp) but with different granularities ("Day" and "Month"). The root node links the objects to the data column "totalPrice", which has been calculated from the columns "grossPrice" and "quantity". Figure 5.3 (right) shows a part of the GraphML file, which gets created after the user clicks the "Done" button on the main screen of the interface. It shows some of the data points in the UTC time format and the non-temporal data, that is linked to the created objects.

There are multiple ways to create the needed TemporalDataset. Another way would be to use the ObjectCatalog to create the configuration more efficiently. In this case, the user would create the first TemporalObject and the child TemporalElement. After the creation, he or she would pin the object to the object catalog and use it multiple times for the configuration, changing only the granularity for the TemporalElement data.

The created walkthrough shows that the requirements, given by the user scenarios, have been fulfilled and the software can be used successfully for the intended purposes.

CHAPTER 6

Discussion

As a basis for this work, the question was asked: How can a visual interface, which includes the design aspects named by Aigner, Miksch, Schumann & Tominski [Aigner et al., 2011], assist the users to import time-oriented data tables? This chapter provides the answer to the asked question and refers to the requirements placed on the interactive visual interface. In the second step, the solution will be critically reflected.

In chapter 1.2 the requirements for the interface were made. The interactive visual interface should be:

- a validated design artefact that provides an interactive visual interface for import time-oriented data from tables,
- considering the design aspects from Aigner, Miksch, Schumann & Tominski [Aigner et al., 2011] &
- usable (avoiding usability problems set by Nielsen [1994a]).

To improve user acceptance, usability inspections were performed. These usability inspections were conducted in several iterations and revealed how the usability increased and if the software can be used in the current state. Although some suggestions for improvement have been suggested by the experts, it has been shown that the software assists the user to do his or her task, which is, to perform the import of time-oriented data from tables (CSV). Special attention was paid to the data-wrangling functionalities, which are based on Wrangler [Kandel et al., 2011]. The experts found these functionalities particularly useful and necessary to implement user scenarios simply and correctly.

Using the visual interface, the user is able to import time-oriented data, supporting various aspects of time-oriented data. The user is able to import data having multiple scopes, arrangements, time primitives and indeterminacies, which were set by user scenarios. For

example, the user is able to import time-oriented data, which consists of time points or intervals. Also, e.g. cyclic and linear data import is possible.

All participants of the usability inspections have managed to implement the required import from the user scenarios. As a result of the import, a TemporalDataset was created, which can be used as a basis for further visualizations in TimeBench. This shows that the software can already be used productively, at least if the, already mentioned, aspects of the time-oriented data have to be considered.

Regarding the main question of the work **"How can a visual interface, which includes the design aspects named by Aigner, Miksch, Schumann & Tominski [Aigner et al., 2011], assist the users to import time-oriented data tables?"**, the software demonstrates a way how such implementation can assist the user to import time-oriented data. TimeBench provides a convenient data structure for storage and link of time-oriented and non time-oriented data, which was the point of reference for the design of the visual interface. The graph representation shows a possibility how the construction of the temporal dataset can be done. At the same time, the representation of the graph is directly linked to the given data structure of the TemporalDataset. This fact simplifies the abstraction level for the user, between the created graph, which is visible for the user, and the actual TemporalDataset data structure in the software. Such a software, that works with raw data must have a functionality to adjust and reshape the data. This can have several reasons. One of the reasons is that the data can only be processed in a certain "shape", as it was in this case. Wrangler shows a way how this can be managed and served as basis for the data-wrangling functionalities in the created visual interface. The user has the possibility to change columns from the tables, to merge them and to create new ones. Similar to Keshif (Chapter 2.2.2), the interface is able to automatically detect data types and can combine different features, which is convenient for the user.

Finally, it can be said, that all the requirements of Chapter 1.2 have been met. As well, the original question of the thesis has also been answered successfully, showing a prototype, how such a visual interface can assist the user importing the time-oriented data tables.

6.1 Critical Reflection

This chapter critically reviews some aspects that relate to work, on the one hand, and implementation, on the other hand.

The first thing that can be critically reflected is the programming language, in which the TimeBench was developed. The Java programming language is loosing popularity due to programming languages designed for developing web applications [Rind, 2017].

Another issue that should be reflected, is the fact that the software was designed to be used exclusively by experts. The reason for this design lies in the requirements made at the beginning of the work. It should be investigated whether the interface can

be designed in a way, that it can also be operated by novices, which are not familiar with the structure of the time-oriented data. To archieve that, the meaning of the TemporalElement, TemporalObjects and TemporalDatasets should be communicated in a language understandable to the user.

To configure a correct import, the interface requires that the time-oriented data be formatted as "yyyy-dd-MM HH:mm:ss". Unfortunately, this is not always possible under certain conditions. For example, if the raw data is in the following format: "ss:mm:HH MM-dd-yyyy". A remedy would be the split function. With the help of this function, the user would be able to separate the individual time fragments (ss, mm, HH, etc.) from each other and then reassemble them in the appropriate format. However, that is relatively time consuming too. To make this significant easier, a function would have to be created which, in the best case, automatically extracts the time from the raw data and parses it into the appropriate format.

An important point, that have to be reflected, is the missing function for saving sessions. Currently, the user has no possibility to save his or her import configuration and to continue with the configuration at a later time. The absence of this feature significantly limits the usability of the interface. A solution suggestion would be to create an additional save button in the interface, which is responsible for saving the session. The function behind the button would have to deposit several things for a later call. First, the user should be called to set a location on the hard disk for the storage data. Once the location has been set, the file must store the CSV file, currently being imported, at the given location. Then the following data has to be stored in a file:

- the path to the CSV file,
- all nodes and edges between the nodes from the main panel with all properties and the position on the panel and
- all nodes and edges between the nodes from the object catalog with all properties and the position in the catalog.

To simplify the later import, the modified CSV file (with new/edited columns) should be saved. Also the modification steps, which can later be used for undo/redo functions, should be saved in the CSV file. The save file could be written to XML and have following shape:

- *the first row*: the path to the CSV file
- *nested object collections*: the collection would include the location of the object on the screen as property (0 - main panel object, 1 - first object in the object catalog, 2 - second object in the object catalog, etc.) and as another property the nested object with all node properties:
  - *node position*,

- *node name*,
- *node type*: Root, TemporalObject, TemporalElementInstant, TemporalElementSpan, TemporalElementInterval,
- *data element object*: with following properties: "column name", "data type" (boolean, numeric, text, date), "filter operator" (<, <=, =, <>, >, >=), "filter operand"
- *data column object*: with following properties: "column name", "granularity" (millisecond, second, minute, hour, day, week month, quarter, year, decade), "filter operand" (<, <=, =, <>, >, >=), and "filter date".

The discussed feature of saving sessions can be seen as a part of the future work, additional to the Chapter 7.

CHAPTER 7

Conclusion

The visualization of the time-oriented data is usually quite complex. There are many aspects of time-oriented data, that play a major role in visualization. For example, time-oriented data can occur in different scales (ordinal, discrete, continuous), scopes (point-based, interval-based), or arrangements (linear, cyclic)[Aigner et al., 2011]. A library, that visualizes time-oriented data and supports time-oriented aspects, like different time primitives, granularities, calenders and (in)determinacy, is called TimeBench. However, the software TimeBench has a specific structure, that processes time-oriented data and also has no import function for raw data that comes from a data table. In other words, an import framework is missing, which prepares the raw data and forms it, that TimeBench can further process and visualize. This resulted in the following research question, which was worked on in this thesis: **How can a visual interface, which includes the design aspects named by Aigner, Miksch, Schumann & Tominski [Aigner et al., 2011], assist the users to import time-oriented data tables?**

To answer this question, a prototype of a visual assistance, for importing time-oriented data tables, was designed and developed. In order to get a clear idea of the functionalities that the software should have, user scenarios were created in the first step. After the 3 user scenarios were created and the exact scope of the software was known, mockups were created as part of the design process. The reason for the creation of the mockups was to show how the users interact with the software, to handle tasks set in the user scenarios. After the user scenarios were created, there was a clear idea of what functionalities the software should encompass, how the user interacts with the software and its approximate appearance. To implement the interactive visual interface, Java 8 was used. To avoid dependencies on various libraries, the entire user interface was created using JavaFX standard objects. In order to configure the import, the user must compose TemporalObjects and TemporalElements using a graph representation. He or she connects temporal data with non-temporal, by using TemporalObjects, which can be generated in a parent-child relationship. After the software was implemented, the last

step was the evaluation. This was carried out in several steps, testing of the software during the implementation and performance of usability inspections, done by experts in 2 iterations. The usability inspections relied on a list of heuristics, created by Nielsen [1994a], which describes usability problems. The inspections were done by 3 experts in separate sessions, lasting between 45 and 60 minutes. After the first two sessions, the software has been improved, based on the results from the already performed usability inspections. In order to control the progress and possible new problems, another session was performed.

The prototyped visual interface and its usability are assisting users, with expertise in time-oriented data, to import time-oriented data tables. The implemented solution has been created to assist the user, to configure an import of raw data into visualization frameworks, like TimeBench. On the one hand, it uses data wrangling methods to form the raw data into a shape that the visualization framework TimeBench can handle, otherwise to enrich the raw data with the users knowledge. The usability has been checked using the heuristics created by Nielsen [1994a]. Through usability inspections, the usability could be brought to an even higher level, taking Nielsen's heuristics into account. The result of the interactive visual interface is a TemporalDataset object, that can be accessed directly in the software library and can be used by other visualizations frameworks.

The prototyped visual interface provides features based on the design aspects named by Aigner, Miksch, Schumann & Tominski [Aigner et al., 2011]. The created interactive visual interface supports multiple design aspects of time-oriented data. The prototype supports different **scopes** (*point-based*, where the information between two points in time is not given and *interval-based*, where a time value stands for a specified interval), **arrangements** (*linear* - each time value has an unique predecessor and successor and *cyclic* - recurring time values appear), **granularities** (*mapping* from time units to smaller or larger ones), **time primitives** (*instant*, *intervals* and *spans*) and **determinacy** (depending on the knowledge of all temporal aspects).

Future work

During the execution of the work, some topics are noticed that can be followed up in the future. In the category **design**, following improvements could be accomplished:

- the object catalog should be renewed. In the current solution, simple screenshots of the panel have been used as previews for the objects. A solution should be created on how the objects in the catalog are best represented so that they are clear and easy to understand and use,
- the main panel, used for the drawing of graphs is static currently. There is no possibility to draw large graphs, since there is no function to move the panel or to resize the panel and/or the panel objects. These functionalities could significantly improve the design and the usability and

- the added data columns and data elements to the object can not be edited. The only way, the user can change the added elements, is to delete the element and add it again with new settings. This can be improved by creating additional windows for editing objects. Also, even if the user can see that the data columns have been linked to the objects, he or she is not able to see to which objects exactly. The mapping should be shown in some way.

There are also a few improvements in the category **data wrangling**:

- in the current solution, it is possible to create 2 data columns with the same column name. This should be fixed in the future work and
- it is not possible to directly take the year column (e.g. 1982) as an instant of granularity year. The only way is to add a suffix "-01-01 00:00:00" to the values to create a valid datetime type in format "yyyy-MM-dd HH:mm:ss", since the prototype only accepts this format. In a future work, it could be investigated how different formats may be automatically recognized and used for import.

Some of the improvements could also improve the **usability** of the prototype:

- when the data columns are linked to an TemporalElement, they turn green in the data table representation. The data columns, which have been adjusted, or used for merge, should also be marked in a appropriate way,
- substring, split, trim and expression language from prefuse [Heer et al., 2005] could be used to enrich the scope of functionalities when adjusting and merging the data columns,
- in a future version, there should be a functionality to save the session and continue to a later time, allowing redo/undo functions to reverse made mistakes and provide data provenance. It should be researched, how the user is able to keep an overview about the done configuration (the done data wrangling, linking the data columns to the TemporalElements, linking TemporalElements to TemporalObjects, etc.).

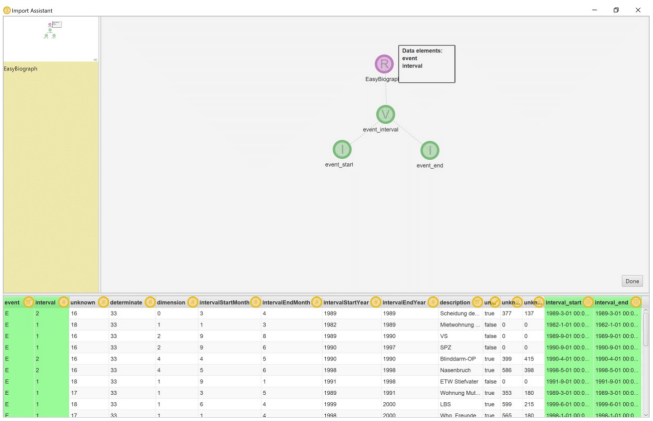
In future works, it can be investigated, how a software for importing raw data could be implemented, so that it can not only be operated by experts, but also by novices, which are not familiar with the structure of the time-oriented data.

Appendix

Appendix A

This appendix shows the sheet, that has been used for the usability inspection in order to present the functionalities of the prototype to the expert.

Visual Assistance for Importing Time-oriented Data Tables



Features:

- Graph representation for easy import of time-oriented data from tables into visualization frameworks
- Support of time-oriented data aspects:
  - Scopes: point-based (instant) and interval-based (interval)
  - Granularities: hours, days, weeks, years, etc.
  - Time primitives: instants, spans, intervals
  - Relations between time primitives: links between intervals or other time primitives
- Separation of data and time elements
- Automatic datatype detection
- Filtering of time elements by minimum and maximum values
- Filtering of non-time elements by text or numerical values
- Object library for reuse of objects
- Adjustment of raw data in columns by using context menus:
  - Increasing/decreasing numerical values
  - Creation of prefixes and suffixes
  - Creation of new datetime column with different starts and granularities
- Merging of multiple columns by using context menus allowing computational operations
- Colour marking of already allocated columns
- The prototype can be imported as library into visualization frameworks and the objects can be accessed via functions
- Export of GraphML XML file

Figure A.1: Prototype functionalities

Appendix B

This appendix shows the usability inspections results.

Test	Index	Issue	Category	Severity Rating	Estimated Effor	Comment	Status
1, 2	1	"Adjust column" context menu should be called in the data table	Usability	low	low		Done
1	2	The contrast of the icons should be adjusted	Design	high	middle		Done
1	3	Wrong edges set in intervals (only to interval end) in GraphML	Result	high	middle		Done
1	4	"Derive" instead of "adjust" which is overwritting the original data	Data wrangling	high	middle		Done
1, 2, 3	5	Changing of data columns & data elements should be possible, not only deletion of all	Usability	middle	high	*Data elements can be deleted separatly now*	Discussed in Thesis
1	6	Disabling of "Done" button, when result not possible	Usability	middle	middle	*Required fields added, closing of dialogues not possible*	Done
1	7	Adding default values to empty fields in dialogues	Usability	middle	middle		Done
1	8	Table preview small	Design	middle	low	*Resizing of data table possible*	Done
1	9	Typo in dataset 3 : audienece ->	Design	low	low		Done
1	10	sort 3rd option after ascending/descending should be dataset order -does not work as expected	Usability	low	high	*Is a standard JavaFX feature*	Minor issue

Test	Index	Issue	Category	Severity Rating	Estimated Effor	*Comment*	Status
1	11	**cells left aligned even though numeric**	Usability	low	high	*Is a standard JavaFX feature*	Minor issue
1	12	**Data table context menu not called if no cell is selected**	Usability	middle	middle		Done
1	13	**Calling of data table context menu on wrong source cell**	Usability	high	low		Done
1	14	**Changing of column data types confusing**	Usability	low	low	*Is only information for the user, the feature is deleted*	Done
1	15	**Display of the currently selected value as an example in the data wrangling windows**	Data wrangling	middle	middle	*The selected column will be shown to the user*	Done
1	16	**adjust dialog the example column value breaks the layout if too long**	Design	low	low	*Is now a text field and no label*	Done
1	17	**"merge two columns" consistency: different texts in popup menu and dialog window**	Design	low	low		Done
1	18	**"merge two columns" the "&" for text concatenation mixed between numeric operations, but quite different from them**	Design	middle	middle		Done
1	19	**"merge two columns" the text field for delimiter should only be enabled for concatenation**	Data wrangling	high	low		Done

Test	Index	Issue	Category	Severity Rating	Estimated Effor	*Comment*	Status
1	20	**"merge two columns" - it is possible to merge columns with an empty operation --> IndexOutOfBoundsException:**	Data wrangling	high	low		Done
1	21	**"merge two columns" it is possible to merge columns into a column with empty name**	Data wrangling	middle	low		Done
1	22	**it is possible to merge columns into a column with same name as existing column**	Data wrangling	middle	middle		Discussed in Thesis
1	23	**diagrams in the catalog are very small (hard to read)**	Design	middle	high		Discussed in Thesis
1	24	**no scrollbars in the diagram area**	Design	middle	high		Discussed in Thesis
1	25	**it is possible to drag a diagram node outside the visible area and never get back**	Usability	middle	middle		Discussed in Thesis
1	26	**after the adjust dialog the column selection jumps to the first**	Usability	low	low		Done
1	27	**it is not possible to directly take the year column (e.g., 1982) as an instant of granularity year (AFAIK only possible by adding string suffix "-1-1 1:0:0") - A format for**	Data wrangling	middle	high		Discussed in Thesis
1	28	**If the input has a string with different format what can I do now? Support dd.MM.yyyy and**	Usability	middle	high		Discussed in Thesis
1	29	**the spinner control to increase and decrease values should also allow keyboard input**	Usability	low	high	*Is a standard JavaFX feature*	Minor issue

Test	Index	Issue	Category	Severity Rating	Estimated Effor	*Comment*	Status
1	30	**"every row is 1 ... starting with ..." is a useful feature but it does not need to be based on a row. Maybe move out into a separate dialog?**	Design	middle	middle	*Is a separate dialog now*	Done
1	31	**checkboxes in adjust dialog make it possible to increase and execute "every row is" at the**	Usability	low	low		Minor issue
1	32	**should there be a background color for columns that are used for a merge (or a derive)**	Usability	low	middle		Discussed in Thesis
1	33	**"every row is 1 ... starting with ..." can be executed without filling the dropdown and date chooser. Apparently with milliseconds and the current datetime. I would**	Usability	low	middle		Minor issue
1	34	**substring, split, trim functions could be useful for the adjust dialog; maybe even the**	Usability	middle	high		Discussed in Thesis
1	35	**add toolbars to make action affordances more visible in addition to popup menus: a vertical toolbar to the table view?**	Design	middle	high		Discussed in Thesis
1	36	**"Add object to catalog" vs. "Add >" should use different wording for different actions for example "pin object to catalog"**	Design	low	low		Done
1	37	**"Add data colum" for Instant is misleading because it suggest you can add more than 1. Trying to add a second is possible in the popup menu but leads to an error**	Design	middle	middle		Done

Test	Index	Issue	Category	Severity Rating	Estimated Effor	*Comment*	Status
1	38	**"Add data colum" is the most relevant action for instant -> I would allow a double click on the instant to choose the data column**	Usability	low	middle		Minor issue
1, 3	39	**it is not possible to review the mapping of a data column to an Instant: you cannot find out if you have e.g., chosen the correct granularity; change is only possible by removing and then**	Design	middle	high		Discussed in Thesis
1	40	**instants have a name "New Instant" which can be changed but is not used for the output. The visual element might be used more valuable to show the name of the data column.**	Result	middle	high	*Unfortunately not possible because of the data structure requirements, the data column names must be unique*	On hold
1	41	**data provenance is missing; in the end it not clear what was done to the data (e.g., incremented values, how merged, suffixed) you need to be careful and take notes manually. neither does it support to save a**	Usability	high	high		Discussed in Thesis
1, 2	42	**Implementation of required fields - especially for the creation of new columns e.g. column names**	Usability	high	middle		Done

Test	Index	Issue	Category	Severity Rating	Estimated Effor *Comment*	Status
1	43	**Add context menu not only to the headers, but also to the table**	Usability	middle	low	Done
1	44	**Derive Column: add prefix and suffix at once does not work**	Usability	middle	low	Done
1	45	**„Add new datetime column“: wrong window name**	Design	low	low	Done
1	46	**Button "Done" not deactivated at program start, even if export not**	Design	middle	low	Done
1	47	**„Remove specific data elements“, “remove all …” etc. should all be deactivated if the function can not be executed (no elements to**	Design	middle	low	Done
2	48	**After sorting the columns, data wranglering was not correct**	Result		*Could not be reproduced*	On hold
2	49	**Save sessions**	Usability	high	high	Discussed in Thesis
2	50	**"Add library object" confusing -> better: "Add catalog object"**	Design	middle	low	Done
2	51	**Position of the sorting arrow covers up the icons in the data**	Design	low	low	Done
2, 3	52	**Redo/Undo functions**	Design	high	high	Discussed in Thesis
2	53	**The user should always be warned when deleting objects (not only when deleting multiple**	Usability	low	low	Done
2	54	**The function for editing nodes names not obvious**	Design	low	middle	Minor issue
2, 3	55	**No documentations created.**	Usability	high	high	Discussed in Thesis
3	56	**Not obvious which data column is linked to TemporalElement**	Design	middle	middle	Discussed in Thesis
3	57	**"merge 2 columns" - merging is only "&" operator. All other operators are not "merging" the**	Design	low	middle	Minor issue

Test	Index	Issue	Category	Severity Rating	Estimated Effor	Comment	Status
3	58	Filter "<>" missing in "Add data	Design	low	low		Done
3	59	Graph representation possibly not the best solution for ALL	Design	low	high		On hold

List of Figures

List of Tables

List of Listings

Bibliography

Aigner, W., Miksch, S., Müller, W., Schumann, H., and Tominski, C. (2007). Visualizing time-oriented data - a systematic view. *Computers and Graphics*, 31(3):401 – 409.

Aigner, W., Miksch, S., Schumann, H., and Tominski, C. (2011). *Visualization of Time-Oriented Data*. Springer, 1st edition.

Bostock, M., Ogievetsky, V., and Heer, J. (2011). D3: Data-driven documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 17(12):2301–2309.

Cooper, A., Reimann, R., and Cronin, D. (2007). *About face 3: the essentials of interaction design*. John Wiley & Sons.

Evans, D. and Taylor, J. (2005). The role of user scenarios as the central piece of the development jigsaw puzzle. In Attewell, J. and Savill-Smith, C., editors, *Mobile Learning Anytime Everywhere: a Book of Papers from MLEARN 2004*, pages 63–66. Learning and Skills Development Agency, London.

Ford, R., Thompson, R., and Thompson, D. (1995). Supporting heterogeneous data import for data visualization. In *Proceedings of the 1995 ACM Symposium on Applied Computing*, SAC '95, pages 81–85, New York, NY, USA. ACM.

Heer, J., Card, S. K., and Landay, J. A. (2005). Prefuse: A toolkit for interactive information visualization. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '05, pages 421–430, New York, NY, USA. ACM.

Ilse Arlt Institut für Soziale Inklusionsforschung (2019). easybiograph. `http://www.easybiograph.com/index.php`. Accessed: 2019-02-17.

Kandel, S., Paepcke, A., Hellerstein, J., and Heer, J. (2011). Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 3363–3372, New York, NY, USA. ACM.

Nielsen, J. (1994a). Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, pages 152–158, New York, NY, USA. ACM.

Nielsen, J. (1994b). Usability inspection methods. In *Conference Companion on Human Factors in Computing Systems*, CHI '94, pages 413–414, New York, NY, USA. ACM.

Pantazos, K., Kuhail, M., Lauesen, S., and Xu, S. (2013). uvis studio: An integrated development environment for visualization. In Wong, P., Kao, D., Hao, M., and Chen, C., editors, *Visualization and Data Analysis 2013*, pages 15–30, United States. SPIE - International Society for Optical Engineering.

Ren, D., Höllerer, T., and Yuan, X. (2014). ivisdesigner: Expressive interactive design of information visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2092–2101.

Rind, A. (2017). A software framework for visual analytics of time-oriented data. Master's thesis, Technische Universität Wien, urn:nbn:at:at-ubtuw:1-106735.

Rind, A., Lammarsch, T., Aigner, W., Alsallakh, B., and Miksch, S. (2013). TimeBench: A data model and software library for visual analytics of time-oriented data. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2247–2256.

Rivero, J. M., Rossi, G., Grigera, J., Burella, J., Luna, E. R., and Gordillo, S. (2010). From mockups to user interface models: An extensible model driven approach. In Daniel, F. and Facca, F. M., editors, *Current Trends in Web Engineering*, pages 13–24, Berlin, Heidelberg. Springer.

Rivero, J. M., Rossi, G., Grigera, J., Robles Luna, E., and Navarro, A. (2011). From interface mockups to web application models. In Bouguettaya, A., Hauswirth, M., and Liu, L., editors, *Web Information System Engineering – WISE 2011*, pages 257–264, Berlin, Heidelberg. Springer.

Satyanarayan, A. and Heer, J. (2014). Lyra: An interactive visualization design environment. *Computer Graphics Forum*, 33(3):351–360.

Schulz, H.-J., Nocke, T., Heitzler, M., and Schumann, H. (2017). A systematic view on data descriptors for the visual analysis of tabular data. *Information Visualization*, 16:232–256.

Stolte, C., Tang, D., and Hanrahan, P. (2008). Polaris: A system for query, analysis, and visualization of multidimensional databases. *Commun. ACM*, 51(11):75–84.

Tableau (2019). Selfservice-Analysen in der Cloud mit Tableau Online. `https://www.tableau.com/de-de/products/cloud-bi`. Accessed: 2019-01-17.

Weaver, C. (2004). Building highly-coordinated visualizations in improvise. In *IEEE Symposium on Information Visualization*, pages 159–166.

Yalcun, M. A., Elmqvist, N., and Bederson, B. B. (2018). Keshif: Rapid and expressive tabular data exploration for novices. *IEEE Transactions on Visualization and Computer Graphics*, 24(8):2339–2352.

Ying, L., Yu-Feng, H., Li-Zhou, F., and Yang, W. (2010). Design and implementation of excel massive data intelligent import system. In *2010 3rd International Conference on Computer Science and Information Technology*, volume 2, pages 328–330.