TU WIEN Informatics

Visualisierung des Flusses von Gesundheitsdaten

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Niclas Arbesser-Rastburg
Matrikelnummer 01625725

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assistant Prof. Dr.in techn. MSc Manuela Waldner

Wien, 7. Juni 2023

Niclas Arbesser-Rastburg	Manuela Waldner

TU WIEN Informatics

Visualizing the Flow of Healthcare Data

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Niclas Arbesser-Rastburg
Registration Number 01625725

to the Faculty of Informatics

at the TU Wien

Advisor: Assistant Prof. Dr.in techn. MSc Manuela Waldner

Vienna, 7th June, 2023

Niclas Arbesser-Rastburg	Manuela Waldner

Erklärung zur Verfassung der Arbeit

Niclas Arbesser-Rastburg

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, 7. Juni 2023

Niclas Arbesser-Rastburg

Danksagung

Zu Beginn möchte ich mich bei meiner Betreuerin Manuela Waldner bedanken, die mir im Laufe dieser Arbeit weit mehr hilfreiches Feedback und Tipps gegeben hat als ich je erhoffen konnte. Ich hätte mir keine unterstützendere Betreuerin wünschen können. Weiters möchte ich mich bei Alexander Degelsegger-Márquez und Lorenz Dolanski-Aghamanoukjan von der Gesundheit Österreich GmbH für die gute Zusammenarbeit bedanken, für den regelmäßigen Austausch und Ideenreichtum, die das Ergebnis dieser Arbeit maßgeblich beeinflusst haben. Besonderer Dank gilt auch meinen Eltern, die mir dieses Studium ermöglicht haben und mich dabei immer unterstützen. Zuletzt möchte ich mich noch bei meinen Freunden bedanken, die meine Studienzeit unvergesslich gemacht haben.

Kurzfassung

Der Fluss medizinischer Daten in einer modernen und vernetzten Gesellschaft zwischen einer Vielzahl von Stakeholdern und Institutionen bildet ein komplexes Netzwerk. Aufgrund seiner Größe und Komplexität ist es für jeden, der Einblick erhalten möchte, schwierig, dieses Netzwerk zu verstehen und herauszufinden, wo persönliche medizinischen Daten letztendlich landen. Visualisierung kann als ein mächtiges Werkzeug eingesetzt werden, um diesen Datenfluss für Experten und normale Menschen zugänglicher und verständlicher zu machen, sowie um Transparenz und Lesbarkeit zu erhöhen. Das Ziel dieser Arbeit besteht darin, Möglichkeiten zur Visualisierung von Teilen dieses Netzwerks zu erkunden und ein Framework bereitzustellen, das die Verbindungen zwischen den verschiedenen Stakeholdern und den ausgetauschten Daten visualisiert. Die Umsetzung erfolgt als Webanwendung, die Interaktivität bietet, um die Benutzererfahrung zu verbessern. Diese Interaktivität wird das Information-Seeking Mantra umsetzen, indem zuerst eine Übersicht gegeben wird und dann das Zoomen, Filtern und Anzeigen von Details auf Abruf ermöglicht wird. Ob die gestellten Anforderungen erfüllt wurden, wird durch Feedback von Experten zu ihren Erfahrungen mit der Anwendung ermittelt.

Abstract

The flow of medical data in a modern and interconnected society between large numbers of stakeholders and institutions forms an intricate network. Due to size and complexity, this network is hard to traverse for anyone wishing to gain insight into where their medical data ends up. Visualisation can be used as a powerful tool, in order to make this flow of data more accessible and easier to understand for experts and everyday people, increasing transparency and readability. This work aims to explore ways to visualise parts of this network and provide a framework that visualises the connections between the different stakeholders in it and the data they exchange. The implementation will be done as a web application which provides interactivity to improve user experience. This interactivity will implement the visual information-seeking mantra, by giving an overview first and allowing zooming, filtering and showing details on demand. Whether the set requirements were met will be determined by gathering feedback from expert users about their experience in using the application.

Contents

CHAPTER 1

Introduction

1.1 Motivation

The healthcare industry is one of the most important sectors in our modern society, as it is responsible for ensuring the well-being of individuals and communities. With the rapid advancement of technology and the increasing use of digital tools and platforms, the healthcare sector is generating vast amounts of data that are critical for improving patient well-being, reducing costs, and ensuring quality care. This data is produced, stored and utilised by the many different stakeholders and institutions involved and travels in a complex network. With a healthcare sector as large as the one in Austria, this network is very complex and intertwined, which poses a challenge for anyone who wishes to gain deeper insights into the data flow or simply get an overview.

Despite the challenge, the ability to get a greater overview of the data flow is vital for decision-makers, in order to be able to make informed choices about policies that impact the healthcare sector and everybody relying on it. Considering the highly sensitive and confidential nature of medical data, being able to track where personal data flow and making this data flow more transparent, is of great interest to the general public.

A tool that provides an easily understandable representation of data flow in the healthcare sector would lead to a better understanding of what happens with medical data. For decision-makers, access to such a tool would grant them deeper insights into the healthcare system's workings and help in making policy decisions. Furthermore, creating a transparent data flow is essential to protecting patient confidentiality and building trust between patients and healthcare providers. For private individuals, access to such a tool would enable them to gain a better overview of the flow of their own personal data and from that allow them to better understand and discuss policy decisions.

An ideal way to get an overview of data is to visualise it. In the medical field, data visualisation is used extensively. When it comes to visualising the healthcare network

and the data flow within, no ideal out-of-the-box solution is available. This work is intended to provide a basis for visualising the data flow in our healthcare system and aid in quickly assessing the data that is still incomplete and will be added over time. The implementation will be done according to the goals set below.

1.2 Aim of the Work

The aim of this work is two-fold. The data as provided by GÖG (Gesundheit Österreich GmbH) needs to be compiled into a database [gö]. For that, an appropriate database schema needs to be worked out. Since the data that will be visualized is not in a complete state, the database must also be easily extensible. Using visualisation and interaction design, a tool will be implemented that will allow a wide range of different users to visually explore and analyse the data from this extensible database.

The tool, as described above, will be provided in the form of a web-based application. This application should enable users to better understand where medical data flows and for what it is used. The specific details of the visualisation, which data to highlight and how to best visually represent the dataflow are also part of this work. Users should be able to easily determine, where data gets transmitted, under which conditions and what the contents of these transmissions are. The tool will provide an overview but also enable users to focus on parts of the data flow and get detailed information on specific elements.

The resulting application will be open-sourced and is going to be used and extended by GÖG. This fact adds a few additional goals and challenges. The need for easy extensibility, to enable GÖG to implement new features and extend existing ones needs to be taken into account during development. Detailed documentation and specification also need to be provided to enable future developers to work with the existing implementation and understand how to implement additional functionality.

1.3 Methodology

To achieve the set goal of structuring data in a meaningful way, the Gesundheit Österreich GmbH provided a starting point in the form of dataflow diagrams showing different parts of the healthcare sector and the dataflow within. The details of that will be discussed in Section 2. Initially, a set of requirements were described and agreed upon together with the colleagues at GÖG. These can be found in Section 4.1. The main task in relation to the provided data was characterising, devising a database schema and transferring the data, from the form it was provided in, into a database. The schema and database had to be well structured and easily understandable to enable GÖG to enter data of their own.

For visualising the data, the current research was reviewed in Section 3.1, in order to determine an expressive, effective and appropriate way to convey the desired information to the user according to the requirements. Appropriate tools, lined out in Section 5, were chosen and used to develop the visualisation tool. Preliminary results were

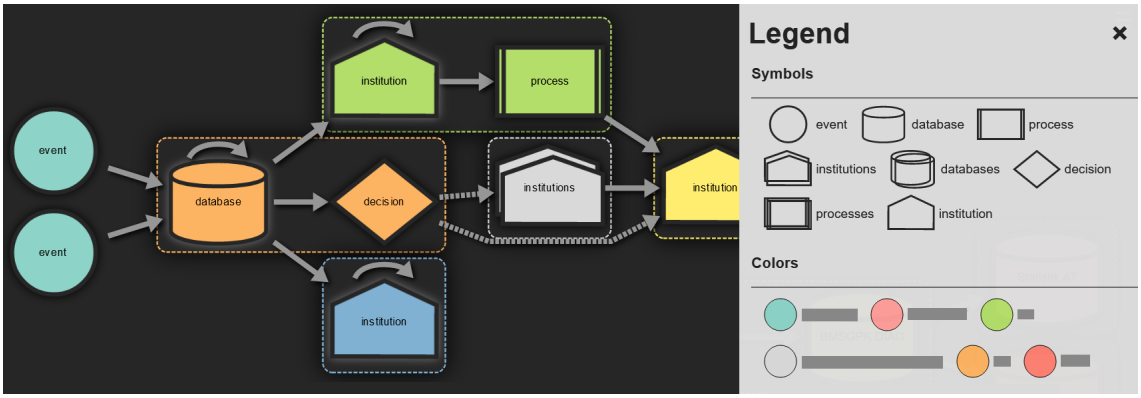

Figure 1.1: Screenshot of the dataflow in a specific scenario, with anonymised node names. The legend explains the meaning of symbols and colours.

regularly presented to the colleagues at GÖG and feedback was incorporated in the following iterations, with the implementation following the concept outlined in Section 4. Extensibility was considered during development and included a database abstraction layer to simplify access and modification of the database. The final step was evaluating the results, where the finished visualisation tool was presented to potential users and maintainers, similar to the walk-through in Section 6.1. Feedback was then gathered on usability and clearness of visualisations, like the one shown in Figure 1.1 and is described in Section 6.3.

CHAPTER 2

Background

Before conceptualising a solution, it is crucial to have a deep understanding of the provided data, the types of users that will utilise the application and the specific tasks that they may want to perform. These are the three corners of the design triangle seen in Figure 2.1, which will help us to expressively, effectively and appropriately represent the given data to users [MA14].

2.1 Data

The Data provided by GÖG came in the form of thirteen PowerPoint slides. These slides depict the data flow in various sections of the healthcare sector, with each slide containing a visualisation similar to Figure 2.2. The slides follow two basic approaches, with the first being scenario-based. The scenario-based dataflow graphs show the data flow that happens after certain events, like a hospital stay or the death of a person. The second

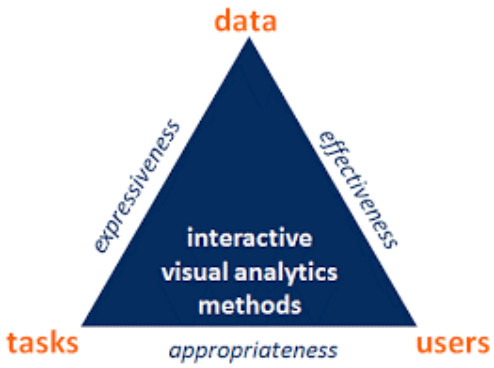

Figure 2.1: The design triangle used for designing interactive data visualisations [MA14].

set of visualisations has a more broad approach in that they depict all data flow between certain entities.

Every visualisation consists of a set of nodes that represent points where data is stored or transformed. Directed edges between the nodes show the direction of data flow and the data being transferred from one node to another. These visualisations were laid out manually with several different node and edge placement approaches. Symbols represent different types of nodes describing processes, institutions, databases and many more. It should be noted though, that the usage of these symbols was not always consistent. Due to the complexity of these graphs, some visualisations, like Figure 2.2, hinted at the desired interactivity by displaying groups of nodes collapsed into larger nodes. These meta-nodes group nodes that logically can be seen as belonging to them. For instance, databases in a hospital could be grouped into a 'hospital' meta-node. Given these hierarchical structures in the discussed slides that are also indicated by the colour coding of nodes, the graph can be described as a compound graph.

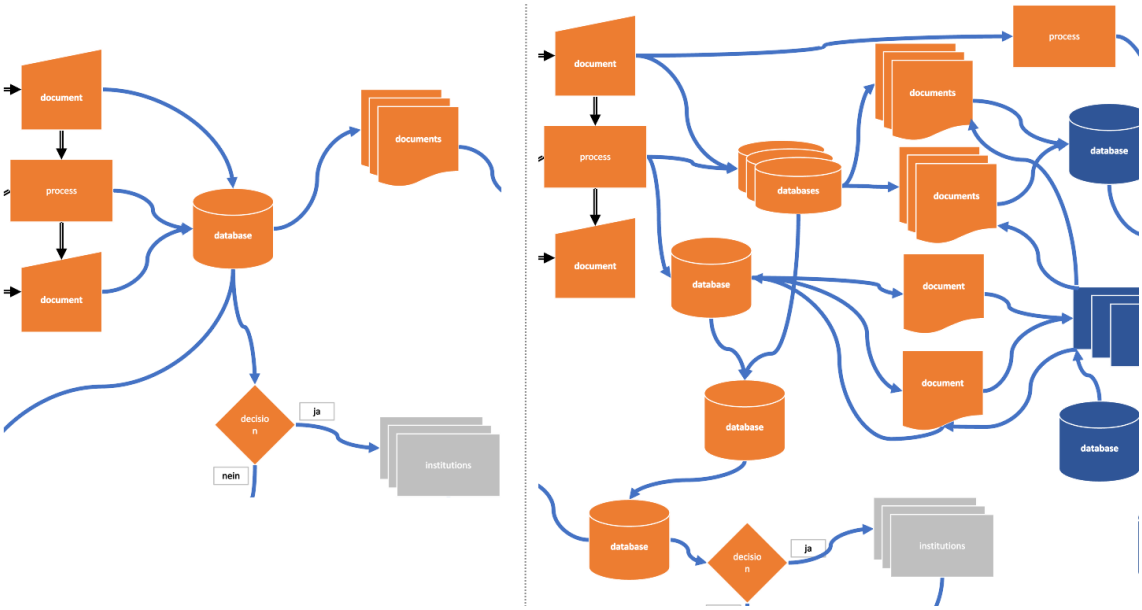

Figure 2.2: Both graphs show the same section of the same scenario. While the right graph shows all nodes within this section, the left graph shows groups of those nodes collapsed into meta-nodes. Nodes in this graph are grouped by the institution they belong to.

A compound graph can be formally described as a graph $C = (G, T)$ with G being $G = (V, E_G)$ and $T = (V, E_T, r)$ being a tree with root r, that both consist of the same set of vertices. As is the case in Figure 2.2, compound graphs can be created by aggregating nodes into new meta nodes, with their respective edges also being aggregated. These new edges and nodes are defined by the nodes and edges contained within them [VLKS+11].

The data contained in these visualisations and accompanying comments were however not complete, which necessitated a very flexible data model and interpretation of what was provided. Many edges did not include concrete data on what was being transferred. A formalisation of what data is necessary for every node and edge had to be created to enable future users to effectively work with the implementation.

Finally, it is important to note, that the data in the provided form was not definitive and not guaranteed to be an entirely accurate depiction of real data flows. In order to prevent readers of this thesis from coming to wrong conclusions when looking at the visualisations, changes have been made to every graph shown in this document. All node names, like ELGA or EMS, have been replaced by their type names, like 'institution' or 'database' [elg] [ems]. Furthermore, when no suitable placeholder is available, text fields are blacked out or replaced by generic 'property: value' descriptions.

2.2 Users

In discussions with the colleagues at GÖG, a number of potential user groups were identified. It is intended that is project is used as the basis for making dataflow in the healthcare sector more transparent to decision-makers, researchers, but also the wider public.

The first user group will take advantage of the database and the database abstraction. They will interact with the application resulting from this bachelor's thesis and will be people familiar with the healthcare sector. Specifically, these will be the data providers at GÖG that will be extending the database and testing out what is possible with the given implementation. They will be interacting with the database out of which the visualisations will be automatically generated.

In a later stage, users will include decision-makers, like politicians and managers, who will take advantage of the visualisation part of this work. The visualisations must therefore be able to effectively convey the complex network that is our healthcare system to the users. This is necessary in order for them to be able to use their learning, to better inform policy decisions. Finally, it is intended to make the tool available to the public. Every person should be able to, easily retrace where personal medical data travels and gets stored based on the data available in the database.

2.3 Tasks

Finally, given the defined user groups, certain tasks need to be defined, that these users want to perform. Knowing these tasks helps in the design of the visualisations and influences the interactivity of the application. The following tasks depending on the user group were therefore laid out.

For the data providers that run and maintain the application, it must be easily possible to add and change data in the database. This is important to enable a workflow that allows

them to quickly identify gaps and inconsistencies in the existing data flow visualisations and fix those in the database. Furthermore, it must also be possible to add new and change existing functionality in the application, so extensibility must be kept in mind during development.

For decision-makers and the general public, it is important that the information is conveyed in a clear manner. They should not be presented with all the information at once but should be presented with an overview of the desired information, They themselves can then decide what is of particular interest to them and can then get specific details. This should help in efficiently communicating the data as it is in the database to the end users, without overwhelming them, for this Shneiderman's Information-seeking mantra offers guidance [Shn96].

Finally, for the general public, it is important to find out what happens to their medical data. It should be possible for them to see what dataflows happen in certain events, like vaccination or surgery. Other than finding out where data ends up and who has access to it, it is also of interest what the transferred data contains, so filtering is also an essential interactivity option.

CHAPTER 3

Related Work

Before conceptualising a solution, it is important to look at what has already been done. Researching related work is important to determine the current state of the art and be able to build on previous research to improve the final results. In my research of research, I initially focused on visualisations related to the medical field: flow diagrams in general and visualisations of dataflow in healthcare networks. Network visualisation itself is a wide-ranging topic, where a lot of interesting research is being done. For instance, Heer et al. presented a powerful tool that aids in visualising large networks [HP14]. It does however lack the directional aspect that is present in our data. Later, I focused on the given input data and how to best visualise compound graphs, group structures and hierarchies.

3.1 Domain Related Research

The most prominent visualisations originating from the medical field are from the various medical imaging technologies [PB13]. However, visualisations are also being used to better understand other types of medical data, detect trends, and changes and gain new insights [Din16]. Different interactive or non-interactive techniques can be used to achieve the desired goals. Kalamaras et al. proposed a solution that incorporates various types of graphs into a large network visualisation where every node represents a distinct graph [KGM+22]. The force-based layout used in this paper was also considered for this project. Another relevant paper treats the symptom progression and treatment of patients as a flow over time [WG11]. Time and the appearance of new symptoms are the X-axis, with patients 'flowing' from left to right. For that purpose, a Sankey diagram was used as shown in Figure 3.1. Those types of diagrams are very useful in conveying flow direction to the viewer. Through interactivity, they let users easily see the flow through the graph. An additional central feature of these types of graphs- that can also be seen utilised in Figure 3.1- it is showing quantities in relation to each other. We can see a large number

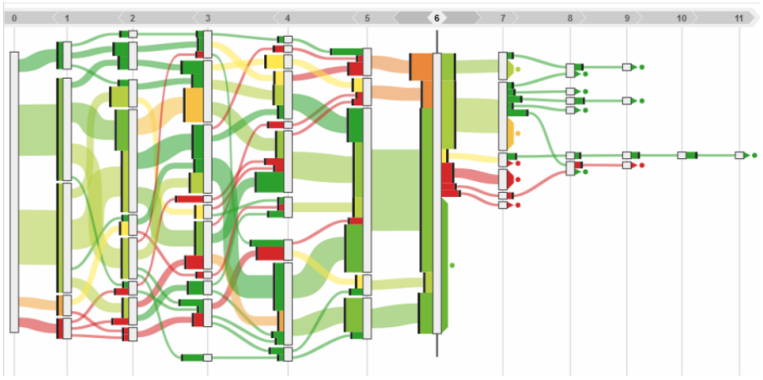

Figure 3.1: Sankey diagram showing patient symptom development 'flowing' thorough time from left to right [WSW+17].

of patients on the right and, with an increasing number of symptoms, the number of patients decreases, with the colours indicating patient outcome. With the main goal of this work being the visualisation of dataflow, Sankey diagrams as a possible solution were considered. However, due to the nature of the input data including hierarchies and the desired interactivity, these types of graphs did not appear to be a promising approach. Furthermore one of the key features of Sankey diagrams, the visualisation of magnitudes was not useful for this project.

Focusing on the healthcare networks themselves and the dataflow within, rather than the contents of the data, various different approaches can be found in the literature. Most research with a focus on visualising healthcare networks focuses on the network itself and the connections that exist within and doesn't emphasise the data flow. A paper by Boddy et al. used data transfers within a hospital as a basis for their network visualisations [BHM+19]. Their goal was to be able to visually examine the complex data set and see user behaviour and interactions, in order to detect redundancies, erratic behaviour and anomalous connections. For that purpose they also used Sankey diagrams, but also different types of force-based layouts, which will be discussed later.

A similar approach was used by Liu et al. [LBW+16], who used visualisations to detect fraud. Recognising that the network of all entities in the healthcare sector is too complex to put into a single visualisation, researchers choose subsets. Figure 3.2 shows the result of their research, which depicts the relationships between a large set of doctors and pharmacies in the context of narcotics prescriptions. Although geospatial aspects also have to be taken into account, several anomalous clusters of doctors and pharmacies are immediately visible, indicating fraud. This visualisation also employs a force-based layout, which is very useful when trying to recognise patterns and anomalies. Force-based layouts are also often used in laying out compound graphs. They unfortunately also have a couple of downsides, which will be discussed below.

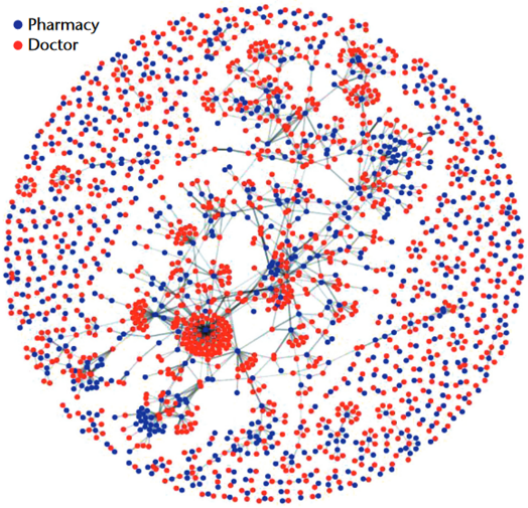

Figure 3.2: Network graph with force-based layout highlighting potentially fraudulent activity [LBW+16].

3.2 Graph Layout

As discussed in section 2.1, the provided datasets can be described as compound graphs. With the goal of clearly visualising the hierarchical structures, while also emphasising the flow direction, there are several papers that can be looked at for guidance.

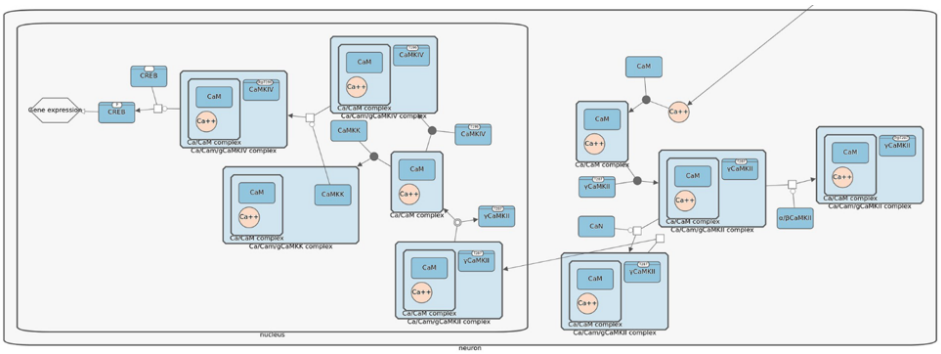

Figure 3.3: Compound graph with constraint based layout [BDOA22].

Compound graphs are, as described in section 2.1, basically two graphs of different structures that share the same nodes [BDOA22]. This complex structure poses a challenge when trying to visualise it in two dimensions. A successful solution is to visualise nodes that share the same parent nodes clustered together. In Figure 3.3, we can see large parent nodes containing smaller nodes which are their child nodes. This approach also supports several levels of hierarchy. Another interesting aspect that can be seen is colour coding: Nodes, which are leaves in the tree structure, are either a darker shade of blue or orange, while parent nodes are coloured in lighter shades of blue. The layout in this paper

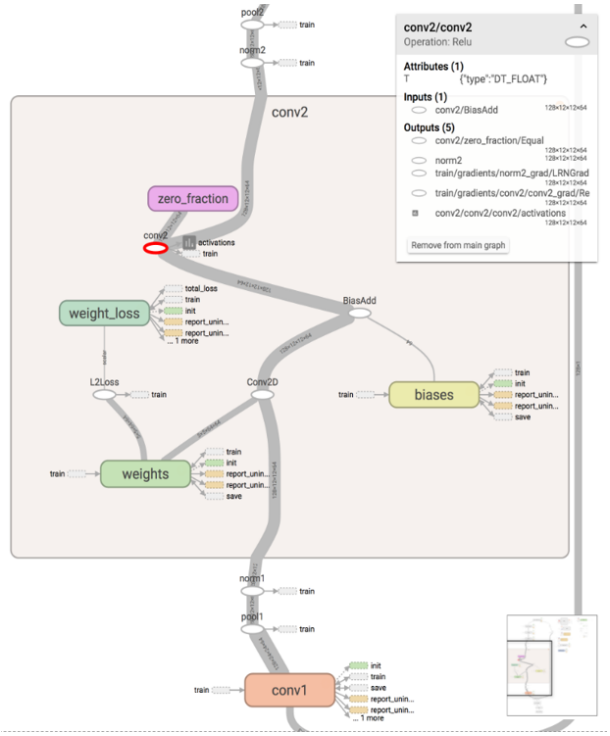

Figure 3.4: Visualisation of dataflow in machine learning models [WSW+17].

was done using fcose, a layout algorithm for compound graphs [BD21]. This approach works well when trying to emphasise group structures. An aspect that is neglected, however, is the visualisation of data flow. Edges and edge directions are clearly visible, but like in the provided PowerPoint slides, it is hard to understand where paths originate and terminate. Therefore, despite producing visually clean layouts, fcose was not suited for this project. The visualisation of groups and the colouring, however, did influence this work.

A different approach for visualising compound graphs was used by Wongsuphasawat et al. in their paper about visualising the dataflow in machine learning models [WSW+17]. Their visualisation clearly shows the data flow from top to bottom. Since machine learning models can be very large, they had to reduce visual complexity in order to be able to present users with an overview. This was achieved by collapsing groups of nodes into their parent nodes. As we can see in Figure 3.4, the orange box with the title 'conv2' is such a parent node that in the current view is expanded to show all nodes and edges contained within. By interacting with that node, it can be collapsed, so that all nodes and edges within are hidden, it would then look similar to the 'conv1' at the bottom of the figure. A further step to reduce visual clutter that was employed by the researchers was to bundle edges and only show edges that connect nodes which are at the same hierarchy level. The techniques used in the referenced paper had a significant impact on this project, particularly in regard to the interactive feature allowing nodes to be

collapsed to manage visual clutter.

CHAPTER 4

Concept

While the current research on related topics was informative, the proposed solutions were not entirely applicable due to not meeting all necessary requirements. The following concept integrates valuable elements from relevant research with the given requirements, forming the foundation for the implementation. Additionally, this concept was vital in the selection of the most suitable tools for the implementation process.

4.1 Requirements

For the final application, the following requirements were set:

1. The implementation has to be extensible. In terms of features and visualisation, it should be possible to make changes and add new functionality easily.
2. Visualisations must be generated automatically out of data stored in a database.
3. It has to be possible to update and change the data in the database.

For the visualisation, the following requirements had to be met:

1. The direction of dataflow must be clearly visible as flowing from one side of the screen to the opposing side.
2. The visualisations should provide the users with all the necessary information to fulfil their tasks, but by incorporating Shneiderman's Information-seeking mantra should not be overwhelming.
3. The visualisations must have a clean layout. Edge crossings should be avoided as much as possible.

4. Group structures have to be clearly visible and compact, with nodes that belong to each other clearly marked as such.

5. The application should support the visual exploration of different scenarios.

4.2 General Idea

Following the design triangle and especially the input data, the choice was made to present the data to users based on real-world scenarios. This had already been done in some of the provided PowerPoint slides, as discussed. The basic idea behind this approach was that the user initially is presented with an overview of all scenarios in the database, shown in Figure 4.1. The user can then choose one of these scenarios and is presented with a visualisation of all associated data. The advantage of that approach is that users are only presented with data that is currently relevant to them. Even with reasonably complex scenarios, users will not be overwhelmed by them. The user then has the option to interact with the displayed graph. All other interactions that are solved with buttons and input fields are located in a menu bar on the top of the window.

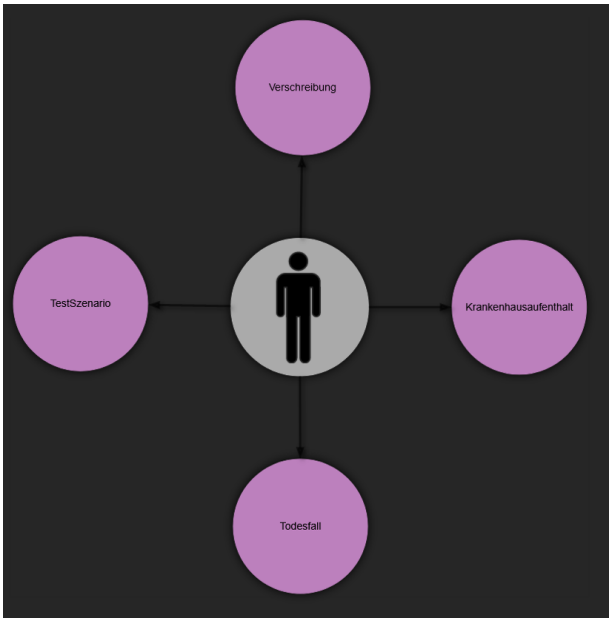

Figure 4.1: As suggested by GÖG, all available scenarios are placed in a circle around a person in the centre.

4.3 Visualisation Design

In the scenario view, the data flow starts on the left with one or multiple events that act as the trigger for the data flow. Edges between the nodes clearly show the direction of the data flow.

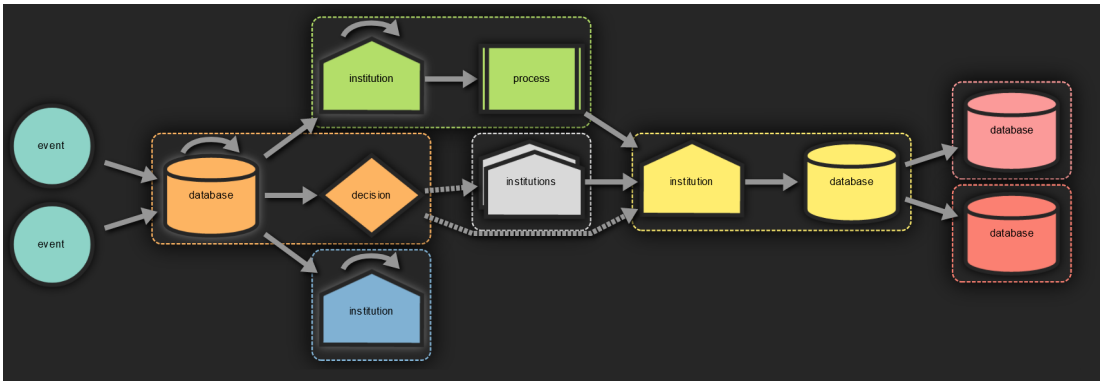

Figure 4.2: Visualisation of the dataflow in a specific scenario.

Nodes are grouped depending on the node group they belong to. In the current implementation, nodes are grouped by the institutions they belong to, like ELGA or GÖG. It is, however, also possible for nodes not to belong to any group. For instance, an event, like an accident for example, often happens independently of any institution. Groups are laid out in a compact way, meaning that nodes that belong to the same group are close to each other. These group structures are further visually enforced in the following two other ways: Nodes are painted in a group-specific colour and groups have a clear border around them in the form of a dotted line that also shares the same colour. This might seem redundant, but is important, because it serves as a visual anchor for when the graph changes as described in Section 4.4.2. Furthermore, it improves accessibility as people with impaired colour vision might have a problem identifying the groups by colour only. As mentioned before, not all nodes belong to a group and therefore obviously are not placed within any marked borders. They are still painted in a default colour, with the exception of event nodes. Due to their special nature, event nodes have their own unique colour, independent of whether they belong to some group or not.

Other than colour, nodes can be identified by the symbol. Every node has a certain type, with an associated symbol. These describe what kind of node is visualised. These range from simple databases to institutions. Many motives are based on ANSI standards[Cha70], with some others being newly created. While symbols like the database are familiar, not all are immediately clear, which is why a legend is provided, that defines all used motives and colours. This legend can be accessed via a button in the menu bar.

Additionally, some nodes, as can be seen in Figure 4.2, have a white drop-shadow. This indicates the possibility of interaction. Nodes with that drop-shadow are meta-nodes that belong to the tree structure of the compound graph. This means, that through interaction they can be expanded to reveal their child nodes.

The concept also allows for special types of edges. For instance, edges originating from decision-type nodes are dotted. This means that the data flow of these edges is not guaranteed. It only happens if the conditions specified in the decision node are met. To give an example, a data transfer might only happen in state-owned hospitals. In the

visualisation that would result in a decision node that contains the query of whether the hospital is public or private with an outgoing conditional edge for when the hospital is public.

Finally, every node has a name, that together with its colour and symbol uniquely identifies the node within the graph. To avoid issues with overly long names, nodes with particularly verbose naming can be assigned abbreviations that will instead be used in the visualisation.

4.4 Interaction Design

Given the complexity of the data and the different tasks users want to perform, an important element in this project, as mentioned before, is interactivity. It is used heavily to make sure that the screen is never overloaded with information, but rather only shows what the user actually wants to see. An important guide to determine what exact functionality would be relevant to implement was Ben Shneiderman's Information-seeking mantra [Shn96].

4.4.1 Overview

Once users have selected a scenario, they will be presented with an overview of the data. By starting with an overview, users are able to get a sense of the big picture and understand how the information is organised. This can help to guide their exploration of the data and make it easier for them to identify patterns and connections. This overview is achieved by aggregating all related child nodes so that only root nodes of hierarchies are displayed. For instance, all nodes in a certain institution will be aggregated into a meta-node representing this institution. Once users have processed this overview, they can move on with exploring the presented information.

4.4.2 Zoom

Initially, it was planned that users could zoom into the graph with the mouse wheel and parent nodes would expand, revealing child all nodes contained within. This would, however, have increased the complexity of the whole visualisation, while a user might only have been interested in a specific set of nodes. Therefore, the decision was made that parent nodes would expand upon clicking on them. This means that nodes that have the aforementioned white drop-shadow can be interacted with using the left mouse button. The node will then disappear and in its place, all direct child nodes and connected edges will appear as seen in Figure 4.3.

Once expanded, nodes can also be collapsed again using the right mouse button. The way this works is that any node with a parent can be interacted with, and then it, together with all nodes that share the same parent, will be replaced with said parent node.

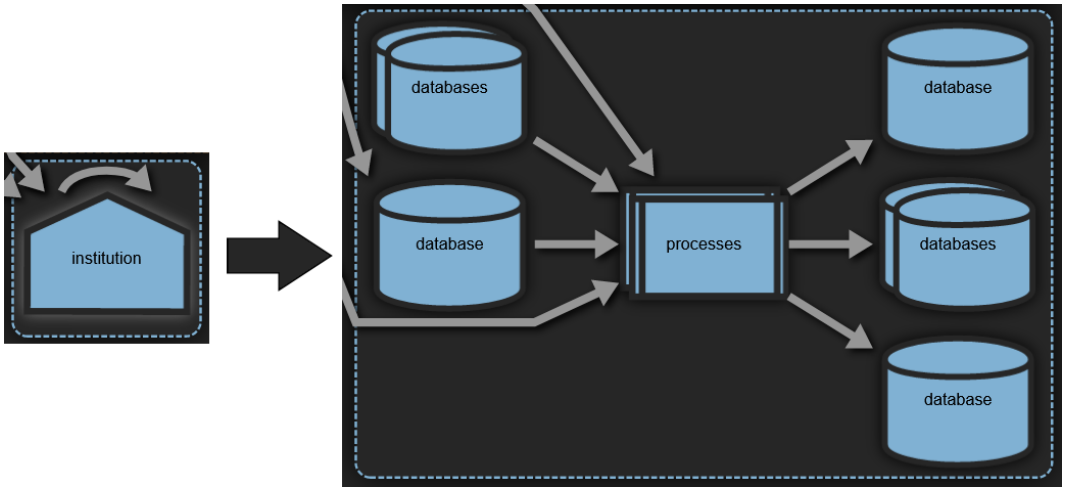

Figure 4.3: Clicking the right institution node reveals all nodes contained within on the right.

Managing collapsing and expanding on a node-to-node basis, depending on the user's wishes, ensures that only relevant data is displayed. It also means that changes to the visualisations are kept at a minimum, which prevents users from getting disoriented during transitions. Zooming in and out using the mouse wheel is of course also possible, but does not affect the contents of the visualisation.

4.4.3 Filter

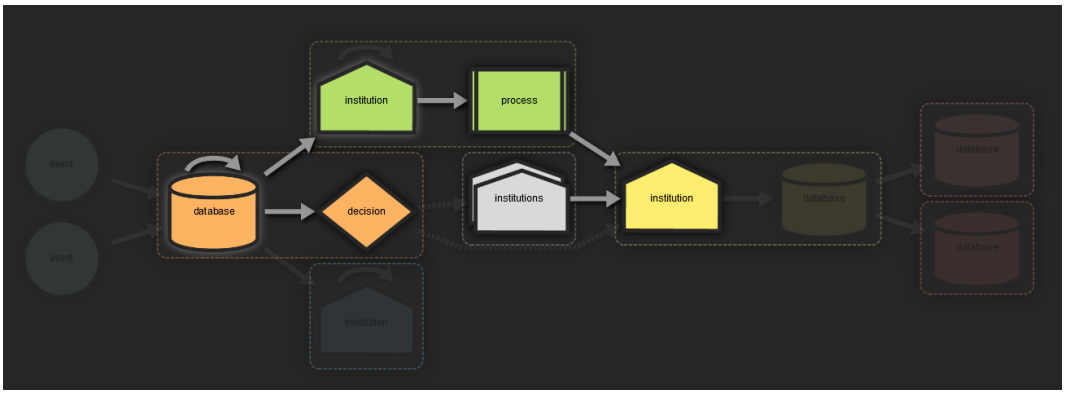

Figure 4.4: Edges were filtered by a search term: all relevant edges and connected nodes are highlighted by greying out all other elements.

Being able to filter the displayed data is another important feature to further refine the displayed data as the user demands. At the top of the screen, users can search either by nodes or edges. A search term can be entered through a text input field. All edges or nodes that do not contain the entered search term are greyed out, thereby highlighting

the relevant parts of the graph. The advantage of this is that the structure of the graph itself does not change, and the filtering is therefore not visually disruptive. Furthermore, the user can still see the search results in the context of the entire graph. For that to work properly, it is important that the greyed-out elements are still visible but clearly distinguishable from the search results, as can be seen in Figure 4.4. Resetting the filter is easily possible via a 'reset filter button'.

4.4.4 Details-on-Demand

Nodes and edges potentially contain a lot of data, much more than could or should be visualised at once. Therefore, details should only be presented to the user when specifically demanded. This is solved here by using text boxes that appear like tool tips and show up when hovering over an edge or node.

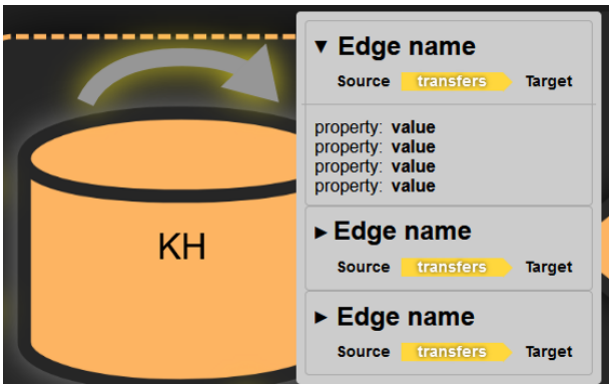

Figure 4.5: Tool-tip of a meta-edge containing three edges, the first one being expanded to reveal contained data.

In the case of nodes, the text box initially only shows the name of the element in bold letters. For edges, the name of the source and target node as well as a colour-coded arrow identifying the type of edge is additionally shown. The reason for this is that one node or edge might contain a larger number of other elements that would quickly fill the screen if displayed with all corresponding data. Therefore, following the details-on-demand -mantra, the contents of specific nodes or edges can be accessed by clicking on the name of the element, which then expands, as shown in Figure 4.5. The element that is currently being hovered over is highlighted by a yellow drop-shadow. As soon as the mouse leaves either the text box or the node or edge, the text box will disappear again.

CHAPTER 5

Implementation

The implementation of the application was divided into various parts in both frontend and backend. To provide a comprehensive understanding of the application's development, the upcoming chapter will delve into the specifics of each component and framework utilised in the implementation process. Figure 5.1 gives an overview of all important elements and components. The graph is divided into software running on the server and on the client machine. To give a quick overview, all data is stored in a neo4j database, images are stored in SVG format in a separate folder [neo]. Everything can be accessed via a spring-boot backend [spr]. On the client side, data is fetched and transformed in databaseService.js. Script.js is the core of the application and gets called upon page load. DotHandler.js uses the fetched data to generate the necessary data used by the visualisation library and keeps the current state over the visualisation. d3-Graphviz then generates the graph on top of which d3, manages interactivity [MBc] [Jac].

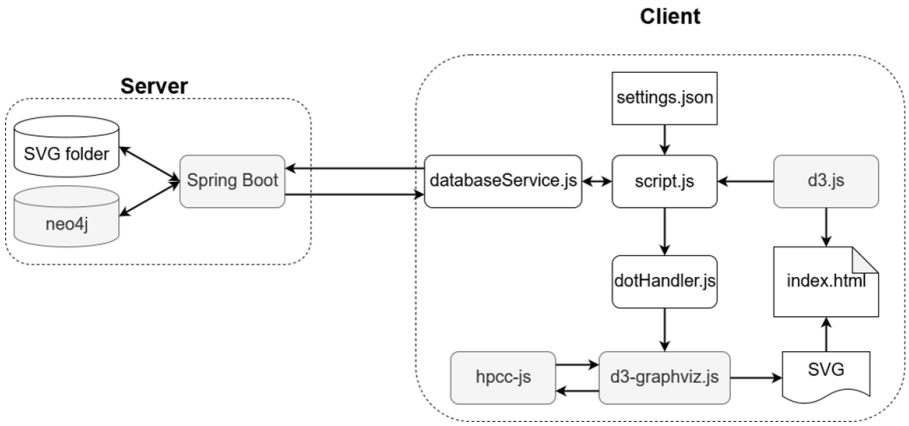

Figure 5.1: Flowchart of important elements on server and client side. Used libraries, frameworks, and other external software are marked as grey.

5.1 Database

In order to be usable for the visualisation, the data from the provided PowerPoint slides had to be transferred into a database. Neo4j is a well-suited database management system for this project because the data provided was in a graph format, representing a network of interconnected nodes. Neo4j is designed for handling complex and highly connected data models, making it a suitable choice for projects with such data structures [neo]. It does not feature any tables or rows. Entities are rather stored as nodes and edges of different types with different attributes. This graph database can then be queried with the cypher query language, which is intuitive to use. For transferring the data from the slides into a neo4j database, all nodes and edges with the little corresponding data that was available were created using cypher queries. All entities have a set of required and optional parameters.

Figure 5.2: Left: JSON representation of a node. Right: JSON representation of an edge, IDs are automatically assigned by the database.

A node must contain at least one label and a name. Labels define the type of symbol that is used in the visualisation but also mark special nodes such as those in Figure 5.2, where we can see an origin-type node. These nodes will always be drawn on the very left. The only other special node label currently in use is ‘parent‘ which is used for drawing the drop-shadow on parent nodes. Looking at Figure 5.2, we can see an object called properties, within it neo4j stores all other user-set variables. As mentioned in Section 4, there are optional variables for an abbreviated node name and a field for relevant information about the node. Database administrators can freely add new variables, which can then be incorporated in the visualisations. As long as the required properties are set the application will work as intended.

Edges similar to nodes have a set of object variables that are required to be set by neo4j and a properties object that contains user-set variables. Some of those are required for the functioning of this implementation. Looking at the right JSON in Figure 5.2, we can see that instead of a ‘labels‘ array we have a ‘type‘ variable that describes what kind of entity it is, with the key difference to node labels being that rather than an array only one type can be set for an edge. In the current implementation, four types of edges are

used:

- transfers: Edges used in the visualisation that signify the transfer of data.
- includes: Used to describe the hierarchical structures. A parent node has an ‘includes‘ edge to its child nodes.
- conditional: Edges originating from a decision node, where data transfer is not guaranteed to happen.
- produces: Edges leading to documents and other content produced by nodes.

Saved within edges are the IDs of the origin and destination node, as well as a properties object, like in nodes. In the current implementation, only edge name and scenario array are required, since for most edges no information was provided. The scenario property defines in which scenarios this edge is relevant.

A running neo4j database can be accessed via HTTP, HTTPS, or the BOLT protocol. All can be accessed via specific ports and by providing a username and password.

5.2 Backend

It is not desirable for every user to directly interact with the database, as that would mean providing them with credentials for the database, which could be used for malicious purposes or accidental changes in the database. For that reason, abstracting the database's access was absolutely necessary. Implementing a backend further provided the opportunity to add endpoints for database administration. For the implementation, the Java-based Spring framework was used with Maven as the build tool. It offers a wide range of capabilities and modules. Specifically for using neo4j, a number of modules are available, the most user-friendly of which is ‘Spring Data Neo4j’. It does, however, not allow for much flexibility in the database, and changes made to it would require changes in the backend. As the data currently contained in the database is incomplete and subject to change, the decision was made to use ‘Neo4j Java Driver’ on which ‘Spring Data Neo4j’ is based. It lacks some convenient tools and quality-of-life features but allows for more flexibility in querying data. Using this driver, the database abstraction was implemented to make RESTful node, scenario, relationship and image endpoints available.

Node-related endpoints offer a wide range of node-related query options. It is possible to query nodes by IDs, names and parent nodes, but also get all direct and indirect parent and child nodes. Adding new nodes to the database is also possible. Most relevant for the visualisations are scenario-related endpoints. Here, all available scenario names can be retrieved, and all data related to a specific scenario can be retrieved. The relationship endpoint is responsible for adding new edges. Here, all relevant edge data has to be provided in addition to either the names or IDs of the source and target node. Finally, the symbols used in the application can also be fetched by name in the form of SVGs.

Figure 5.3: A relationship object containing two nodes and edge-related data.

Depending on the specific endpoints, data is passed either in the form of discrete values, like IDs and names and arrays of those, or as objects and maps of objects. There are two types of objects used. The first are node objects containing all data as described in the previous section. The second are relationship objects. These basically represent edges, but rather than storing the source and target nodes as IDs within, relationship nodes store both node objects to simplify the backend response. In Figure 5.3, we can see a relationship object that could be passed to the backend to add a new relationship to the database. At the top, we see the edge-related data and within source and target, we see node objects containing ID and name. For adding new edges, the node object do not have to contain all node data but only either the IDs of both nodes or the names of both nodes.

5.3 Data transformation

As described, the data from the server is received by the client mostly in the form of node and relationship objects. For the visualisation of a scenario, the backend is queried for the specified scenario, and the response comes in the form of an array of relationship objects. The d3.js library, however, cannot use the data in this form. To fit d3's requirements, the data is transformed into an array of nodes and an array of edges and the required properties set.

5.4 Graphviz

For the visualisation, the d3-graphviz library was chosen as it provided layouts with consistent quality and came with a large set of features. As the name implies, the library used both Graphviz and d3. Graphviz is a graph visualisation library developed by AT&T in 1991 and is written in C++. Through the hpcc-js javascript library, a Web Assembly-ported version of Graphviz can be used on the web [hs]. The aforementioned d3-graphviz library then connects that with the powerful d3 library for interactive data visualisation.

```
digraph G{
graph [ layout = "neato", fontsize = "90", bgcolor="#262626"]
node [ fontname = "Helvetica,Arial,sans-serif", shape = "circle",
    color = "yellow", style = "filled",  width="2.5"]
edge [ len="3.7", color = "#969696", style = "bold", penwidth = "7",  ]


0 [label ="center", color = "#AAAAAA"]
1 [label ="Scenario 1", color = "#BC80BD"]
2 [label ="Scenario 2", color = "#BC80BD"]
3 [label ="Scenario 3", color = "#BC80BD"]
4 [label ="Scenario 4", color = "#BC80BD"]
5 [label ="Scenario 5", color = "#BC80BD"]

0 -> 1 [id = "1"]
0 -> 2 [id = "2"]
0 -> 3 [id = "3"]
0 -> 4 [id = "4"]
0 -> 5 [id = "5"]
}
```

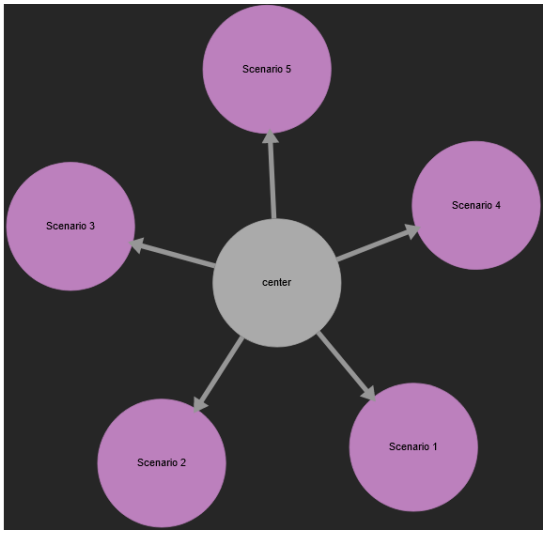

Figure 5.4: Left: dot language string first defines settings for graph, node and edges, then defined individual nodes and finally individual edges; Right: graph produced out of dot language string.

All Graphviz-related tasks are handled by the dotHandler class. For initialisation, an object of that class needs to be provided with the HTML element the graph should be attached to. The object is then responsible for keeping the state of the visualisation. It stores all nodes, edges, and settings, which can all be updated to change the visualisation. When given the command to render the graph, the object must transform all given data into a string that is then passed to d3-graphviz. This string is in the dot language, Graphviz's own language for textually defining a graph. A simple example can be seen in Figure 5.4. The aforementioned settings used by the dotHandler object refer to Graphviz settings that can define things like node size or edge length. When given the command to render the graph, d3-graphviz passes the string to the hpcc library which then returns the graph as a large SVG, which is then attached to the DOM. Transitions between the old and newly generated graph are handled by d3-graphviz to be as little visually disruptive as possible.

5.5 Visual output

The graph that is rendered on screen, provides all the interactivity as outlined in the concept. For that, different types of events are attached to elements of the graph. These events are attached to the graph directly after rendering using d3.

For collapsing and expanding nodes, mouse click events are used. The ID of the clicked node is then looked up in a map containing all parent-child relationships. If the left mouse button was clicked, and the clicked node is a parent node, it and all its edges will be removed from the dotHandler instance. Instead, all child nodes and their edges will be added and rendered. If the right mouse button was clicked and the specified node has a parent, all nodes that share the same parent will be removed from the dotHandler object, and the parent node will be added. All edges attached to the removed nodes will instead be attached to the newly added parent node. To achieve a smooth transition between the old and new visualisation, d3-graphviz stores both states of the graph and moves already existing elements to their new positions, and fades in new elements.

The implementation of the filter option was quite simple, yet effective. At the top of the window is a text input with a drop-down menu. Here, the user can choose whether to filter by node or edge and can then enter a search term. Ideally, it would be great to offer additional options such as filtering nodes by their legal basis. However, due to the lack of data, the filter option currently searches only the details' property of nodes and the content property of edges. For visualising the search results, the implementation uses d3. Here, the opacity of all nodes or edges that do not contain the data specified in the search term is reduced.

Node- and edge-related details are shown using in the form of a tool-tip-like text box. This is achieved using 'mouseover' and 'mouseleave' events. When the mouse is over a node or edge, all data related to the element is retrieved by ID and visualised as a 'details' HTML element that can be extended and collapsed.

5.6 Extensibility

As required by the colleagues at GÖG, the implementation offers a lot of customizability. Effort was put into enabling future developers to easily make additions and changes to the code. To allow major changes to the database design without requiring extensive changes to the backend, the different properties of nodes and edges are not hard-coded, and the Java neo4j driver allows for secure interaction with the database. A settings file enables changes to the way interactivity works. It enables administrators to make changes to the visual appearance of the web application, as well as Graphviz settings and default behaviours. Many of the things outlined in this work, like node colouring based on group and even layout settings, can actually be changed if desired. Interactivity is handled by functions that initialise all necessary events. These functions are passed to the dotHandler object. Changing or replacing these functions is easily possible. Changing the view or switching back to the scenario overview is also simple to change, the currently

used dotHandler object just needs to be destroyed and a new one initialised. These and all other possibilities for change and extension are explained in great detail in the extensive documentation of this project. The specification for the backend was done by using the Javadoc maven module, which creates a web page out of all Javadoc in the project. For the frontend specification, Natural Docs, also generated a web page, resulting in an easily navigable and searchable specification [Val].

CHAPTER 6

Results

This chapter will delve into the results of the implementation process, providing insight into how the application functions in practice. To achieve this, a brief walk-through of an example user interaction with the application will be presented and explained. Following this, the visualisations generated by the implementation will be compared to the original PowerPoint slides. Additionally, feedback from target users at GÖG will be discussed, providing valuable insight into the application's usability and potential areas for improvement.

6.1 Walk-Through

User interaction with the application starts with the scenario overview as seen in Figure 4.1. After the user selects one of the scenarios represented by the purple nodes, the dataflow visualisation for the selected scenario is presented. This looks similar to Figure 6.1, with the graph centred in the middle and the menu bar on top.

In Figure 6.1, we can see an overview of the entire graph and the mouse hovering over an edge indicated by the yellow drop-shadow. The tool-tip box reveals that the edge in question actually contains two edges bundled into one meta-edge. The names of the edges, as well as the nodes they connect, are shown.

In the next step, the user might want to filter the graph by a search term, which in this example is a database identifier. In Figure 6.2 we can see the search term, in red, at the top of the input field. After hitting the 'filter' button, all nodes and edges are greyed out, except for those containing the identifier. In Figure 6.2 we can clearly see in which database it first appears and where it gets transmitted, and if we look at the details of one of the highlighted elements, we can see the database identifier (here marked in red).

After hitting the 'reset filter' button to return to the normal view, a user might wonder what the different symbols or colours mean. Looking at Figure 6.2, we can see a button in

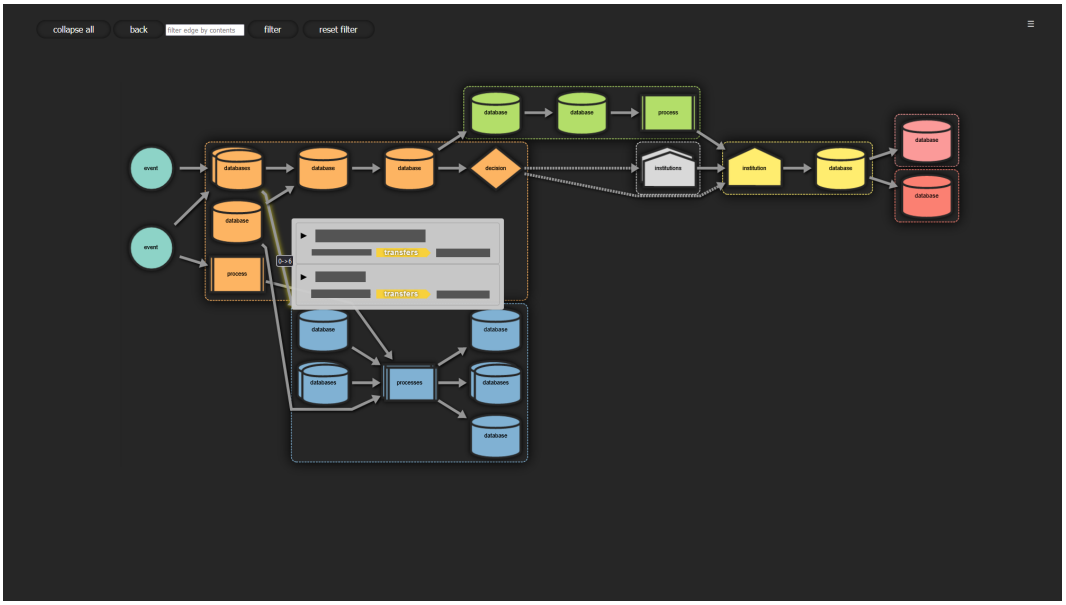

Figure 6.1: Web page of dataflow visualisation with menu bar at the top and graph in the middle. Hovering reveals info about the specified edge.

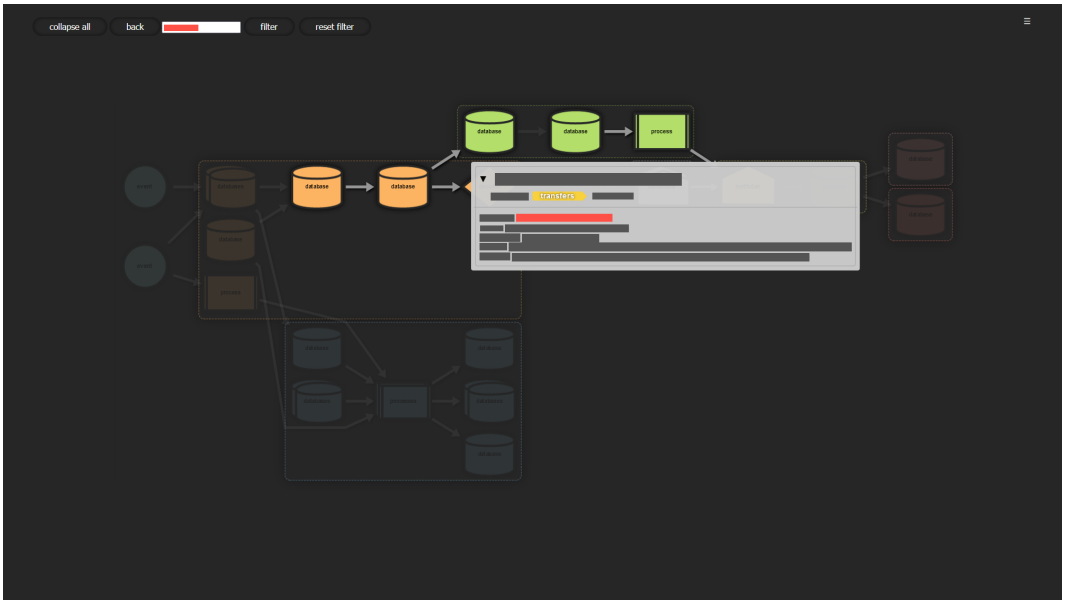

Figure 6.2: Graph filtered by search term in red, search term can be found in the highlighted edge.

the top right corner, and interacting with it reveals the legend. The legend appears on the right side of the screen in Figure 6.3 and lists all symbols with their respective meaning and all colours and their significance, in this image they describe different institutions.

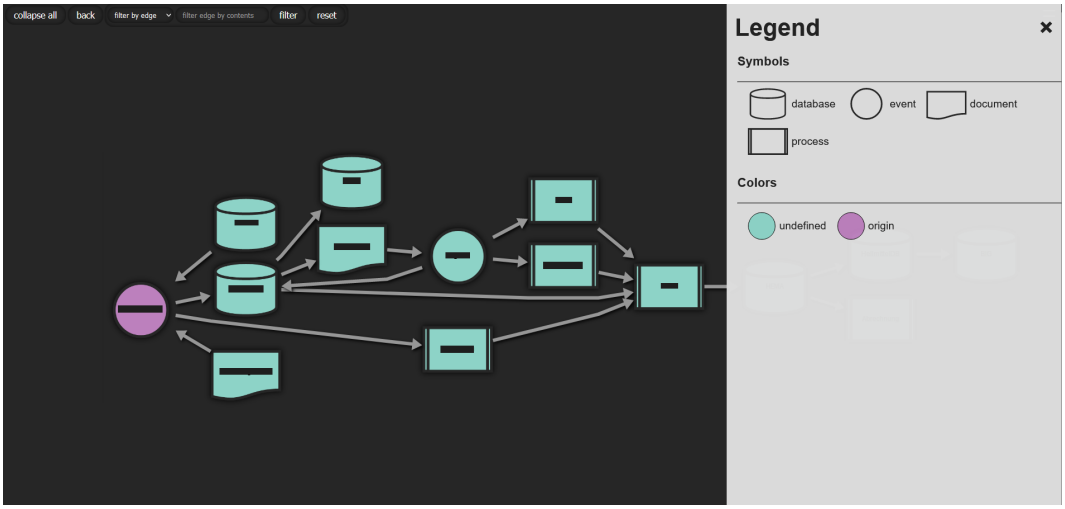

Figure 6.3: Legend of a scenario where very little data was available, explaining all used symbols and colours.

6.2 Comparison to input data

As discussed, all provided data that was used for the application was contained in a set of PowerPoint slides as graphs. These graphs were laid out manually. Generally, the origin of dataflow was placed on the top left. However, no clear direction of flow was visible, and many edges crossed over each other. Another important element of the slides was the data contained within nodes and edges. The quality of information was quite inconsistent, and some of the nodes and edges came with no data at all. Upon inquiry on specific details of dataflows, the colleagues at GÖG came to the realisation, that some scenarios were much more complex than their initial visualisations had suggested. This highlighted one of the many advantages of this formal approach, as getting the necessary data for the application to function, required the data providers to really delve into their data sources, leading them to discoveries they might otherwise not have made.

When comparing the top graph taken from the slides in Figure 6.4 to the graph below depicting the same scenario, but produced by the application, we can see a number of key differences. As required, the data now consistently flows in one direction. Another key difference is the general design: a lot of time was invested in making the visualisation look modern, ensuring that visually pleasing colours are assigned. A final notable difference can be seen in the form of the legend that is at the bottom of the top graph, due to not always being relevant the legend in the application is not visible by default.

6.3 Feedback

Throughout the development process, GÖG provided regular feedback on the current state of the application in meetings that occurred approximately every two weeks. One

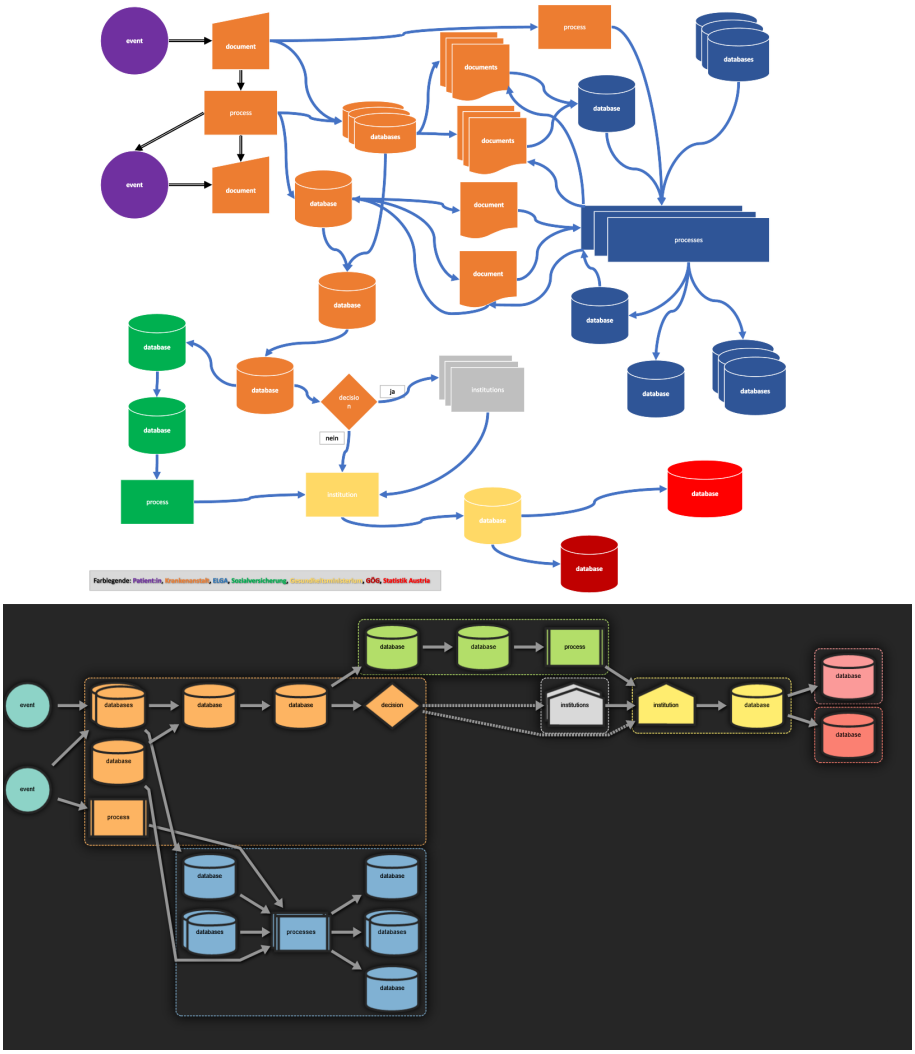

Figure 6.4: Top image shows one of the initially provided graphs, the bottom image shows the automatically generated graph of the same data fully expanded.

of the earliest topics of discussion was the choice of layout and visualisation, as early results did not meet all requirements. As development progressed, feedback in subsequent meetings primarily focused on improving the aesthetic aspects of the application, as well as addressing specific user needs. These regular meetings ensured, that GÖG was kept informed on the current progress and could provide feedback as soon as new features and changes were implemented. These feedback discussions helped inform many design and development decisions.

The code repository, together with the database and documentation of both were handed over to GÖG in the middle of March. The returned feedback was very positive. When comparing the new automatically generated visualisations to the original PowerPoint

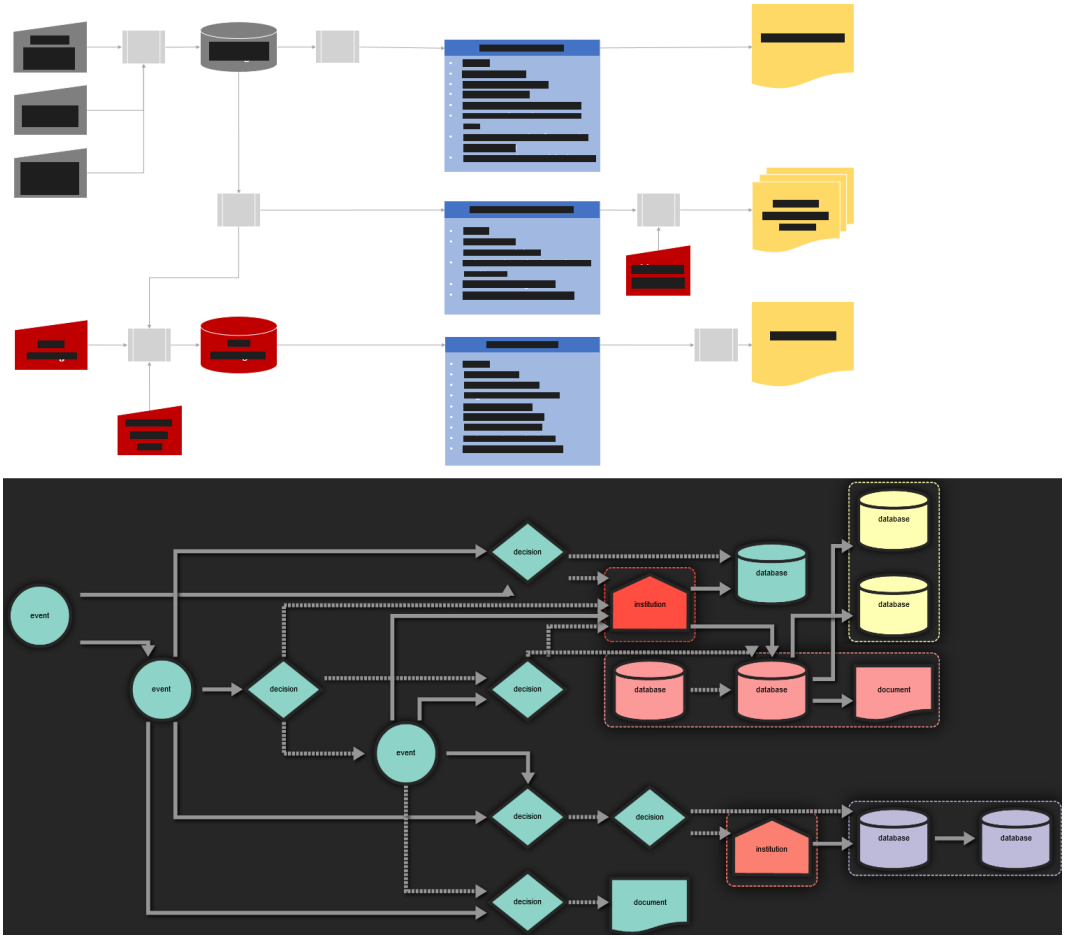

Figure 6.5: Another comparisonwhere the top image shows one of the initially provided graphs, the bottom image shows the automatically generated graph of the same data with orthagonal edge layout.

slides, significant improvements in comprehensibility were noted. These generated visual representations also aid in identifying gaps within the input data.

One of the main goals was to provide an implementation that can easily be expanded and changed. As they commented, it was more important to them to receive a well-structured and documented implementation rather than a perfect but unnecessarily complex and sparsely documented solution. Due to time constraints, GÖG was yet able to implement features. Given the extensive documentation of both frontend and backend as well as the clear separation of responsibilities between software components, they feel confident that making changes of their own will not pose a problem.

Finally, the colleagues of GÖG noted their confidence that this project has laid a solid basis for helping them convey the complex data to clients with no in-depth knowledge of the topic. Despite not having been able to test the application in use, they feel that the

result of this project has more than met their expectations.

CHAPTER 7

Conclusion

The primary goal of this work was to provide the Gesundheit Österreich GmbH with a tool that would help them to effectively communicate the complex data flow in the healthcare system in a clear and intuitive way. To achieve this goal, all graphs shown to users are generated automatically, with new graphs easily added by data providers simply by adding new data to the database. As it is planned to extend and possibly change the functionality of the application, many settings and behaviours can easily be changed, or replaced as needed. By creating a tool that is both flexible and user-friendly, this work will help to streamline the communication of healthcare data, enabling more efficient decision-making and analysis.

Communication with GÖG throughout the implementation process was consistently helpful, with productive meetings that fostered the exchange of new ideas and improvements. Although some data missing from the original slides was provided, much of the required data could not be provided as this project is just the starting point for the data providers to complete the dataflow models.

The early stages of implementation proved challenging due to an initial choice of tools that were not suitable for the project's requirements. Several different graph layout libraries were utilised but produced unsatisfactory results. Initial attempts to use a force-based layout were inappropriate, leading to the exploration of various algorithms for directed acyclic and compound graphs. Ultimately, Graphviz was chosen as the best solution. While being a good fit, given the set requirements, Graphviz also had a couple of downsides. It, unfortunately, does not only produce the graph's layout but renders the entire graph as an SVG, which takes a bit of control away from the developer. This also means that all SVGs used as node shapes have to be provided before rendering, which adds unnecessary complexity. It would be preferable if the library only returned the layout and allow the developer to have full control of the visualisation. Graphviz, however, remains a solid choice, as it provided a wide range of easy-to-use customisation options, far more than could have been implemented in the course of this project. The

library also has very good documentation with examples and explanations. Initially, it was also intended to use vue.js as a frontend framework, but due to the mentioned frequent changes of used libraries in the beginning and not many use cases for vue.js' capabilities, it was deemed unnecessary and removed from the project. Looking back it might have been a good idea to keep vue.js as part of the project, to manage button and input field components.

As described in Section 6.3, the degree to which the goals have been met is hard to precisely assess since the colleagues at GÖG were not yet able to test all features and requirements of the provided application. In terms of visual output, the result offers improved readability over the initial visual representation of the data.

This work is intended as the basis for future enhancement, with many aspects of the current implementation offering interesting topics for future work. Currently, only two views are implemented, one showing an overview of all scenarios in the database and one showing a specific scenario, but many more interesting visualisations with the given data would be possible. Filtering options could also be vastly expanded, for instance, filtering by who gets data anonymised or not anonymised or which transfers happen because of a given law could easily be implemented. The main requirement for this would be data of consistent quality. In the backend, once every type of node and edge and all possible properties are definitively specified, the database connection should be handled by spring neo4j which is more convenient to use for developers. Finally, as stated above, Graphviz and d3-graphviz, as well as d3 itself, offer a wide range of possibilities in terms of visualisation, transition and interaction, that are currently not being leveraged to the fullest extent.

Bibliography

[BD21] Hasan Balci and Ugur Dogrusoz. fcose: a fast compound graph layout algorithm with constraint support. IEEE Transactions on Visualization and Computer Graphics, 2021.

[BDOA22] Hasan Balci, Ugur Dogrusoz, Yusuf Ziya Ozgul, and Perman Atayev. Syblars: A web service for laying out, rendering and mining biological maps in sbgn, sbml and more. PLOS Computational Biology, 18(11):e1010635, 2022.

[BHM+19] Aaron Boddy, William Hurst, Michael Mackay, Abdennour El Rhalibi, Thar Baker, and Casimiro A Curbelo Montañez. An investigation into healthcare-data patterns. Future Internet, 11(2):30, 2019.

[Cha70] Ned Chapin. Flowcharting with the ansi standard: A tutorial. ACM Computing Surveys (CSUR), 2(2):119–146, 1970.

[Din16] Ivo D Dinov. Methodological challenges and analytic opportunities for modeling and interpreting big healthcare data. Gigascience, 5(1):s13742–016, 2016.

[elg] ELGA elektronische gesundheitsakte. `https://www.elga.gv.at/`. Accessed: 2023-06-06.

[ems] EMS elektronische gesundheitsakte. `https://datenplattform-covid.goeg.at/EMS`. Accessed: 2023-06-06.

[gö] GÖG gesundheit Österreich gmbh. `https://goeg.at/`. Accessed: 2023-06-06.

[HP14] Jeffrey Heer and Adam Perer. Orion: A system for modeling, transformation and visualization of multidimensional heterogeneous networks. Information Visualization, 13(2):111–133, 2014.

[hs] hpcc systems. @hpcc-js/wasm hpcc systems wasm collection. `https://hpcc-systems.github.io/hpcc-js-wasm/`. Accessed: 2023-06-06.

[Jac] Magnus Jacobsson. d3-graphviz graphviz dot rendering and animated transitions using d3. https://github.com/magjac/d3-graphviz. Accessed: 2023-10-04.

[KGM+22] Ilias Kalamaras, Konstantinos Glykos, Vasilis Megalooikonomou, Konstantinos Votis, and Dimitrios Tzovaras. Graph-based visualization of sensitive medical data. Multimedia Tools and Applications, 81(1):209–236, 2022.

[LBW+16] Juan Liu, Eric Bier, Aaron Wilson, John Alexis Guerra-Gomez, Tomonori Honda, Kumar Sricharan, Leilani Gilpin, and Daniel Davies. Graph analysis for detecting fraud, waste, and abuse in healthcare data. Ai Magazine, 37(2):33–46, 2016.

[MA14] Silvia Miksch and Wolfgang Aigner. A matter of time: Applying a data–users–tasks design triangle to visual analytics of time-oriented data. Computers & Graphics, 38:286–290, 2014.

[MBc] Jeffrey Heer Vadim Ogievetsky Mike Bostock, Jason Davies and community. D3.js data driven documents. https://d3js.org/. Accessed: 2023-10-04.

[neo] neo4j graph database. https://neo4j.com/. Accessed: 2023-10-04.

[PB13] Bernhard Preim and Charl P Botha. Visual computing for medicine: theory, algorithms, and applications. Newnes, 2013.

[Shn96] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In Proceedings 1996 IEEE symposium on visual languages, pages 336–343. IEEE, 1996.

[spr] Spring Boot. https://spring.io/projects/spring-boot. Accessed: 2023-06-06.

[Val] Greg Valure. Natural Docs readable source code documentation. https://www.naturaldocs.org/. Accessed: 2023-10-04.

[VLKS+11] Tatiana Von Landesberger, Arjan Kuijper, Tobias Schreck, Jörn Kohlhammer, Jarke J van Wijk, J-D Fekete, and Dieter W Fellner. Visual analysis of large graphs: state-of-the-art and future research challenges. In Computer graphics forum, volume 30, pages 1719–1749. Wiley Online Library, 2011.

[WG11] Krist Wongsuphasawat and David Gotz. Outflow: Visualizing patient flow by symptoms and outcome. In IEEE VisWeek Workshop on Visual Analytics in Healthcare, Providence, Rhode Island, USA, pages 25–28. American Medical Informatics Association, 2011.

[WSW+17] Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mane, Doug Fritz, Dilip Krishnan, Fernanda B Viégas, and

Martin Wattenberg. Visualizing dataflow graphs of deep learning models in tensorflow. IEEE transactions on visualization and computer graphics, 24(1):1–12, 2017.