

Visualization of multi-layered data in node networks

Jacob Rosensköld and Magnus Sunesson

DEPARTMENT OF DESIGN SCIENCES | FACULTY OF ENGINEERING LTH
LUND UNIVERSITY 2021

MASTER THESIS



Visualization of multi-layered data in node networks

Jacob Rosensköld & Magnus Sunesson



LUND
UNIVERSITY

Visualization of multi-layered data in node networks

Copyright © 2021 Jacob Rosensköld & Magnus Sunesson

Published by
Department of Design Sciences
Faculty of Engineering LTH, Lund University
P.O. Box 118, SE-221 00 Lund, Sweden

Subject: Interaction Design (MAMM01)
Supervisor: Kirsten Rasmus-Gröhn
Co-supervisor: Andrew Basterfield and Jianhua Cao at
Sinch
Examiner: Günter Alce

Abstract

Visualization is the discipline of representing information with visual elements. Data structures can benefit from visualization, enabling people deeper understanding of the structures. The more complex the data structure, the harder it can be to keep track of different relations between different objects in the structure without a good visual representation. The aim of this thesis was to create a tool for visualizing a multi-layered data structure for a company called Sinch. After a literature study on, among other subjects, visualization, graph drawing, and interaction design, a tool that is an interactive node network was developed. Using force-directed graph drawing algorithms for finding optimal node placement, simulated annealing algorithms for label placement, and other techniques, a single-page web application was built. The tool is presented in this thesis with pictures and in-depth descriptions of how the interaction is to work, as well as the thought processes behind the different design choices.

Keywords: Visualization, Interaction Design, Node networks, Graphs

Sammanfattning

Visualisering är läran om hur man representerar information med visuella element. Datastrukturer kan tjäna på att visualiseras, då det kan öka folks förståelse för strukturen. Ju mer komplex en datastruktur är, desto svårare är det att hålla reda på de olika relationerna mellan olika objekt i strukturen utan en bra visualisering. Målet med detta examensarbete var att skapa ett verktyg för visualisering av en datastruktur med flera nivåer åt företaget Sinch. Efter en litteraturstudie på bland annat visualisering, grafitning och interaktionsdesign skapades ett verktyg som är ett interaktivt nodnätverk. Genom att använda algoritmer som force-directed grafitning för att finna optimal nodplacering, simulerad glödning för placering av nodetiketter skapades en enkelsidig webbapplikation. Verktöget presenteras i det här examensarbetet med bilder och djupgående beskrivningar av hur interaktionen är tänkt att fungera, såväl som tanken bakom olika design beslut.

Nyckelord: Visualisering, Data, Interaktionsdesign, Nätverk, Grafer

Acknowledgements

We want to thank our supervisor Kirsten Rasmus-Gröhn from LTH for her valuable feedback on writing a thesis. We also want to thank our supervisors from Sinch, Andrew Basterfield and Jianhua Cao, both of whom helped us in the process from start to finish. We want to thank our examiner Günter Alce from LTH for his help. Finally we want to thank our family and friends for their support.

Lund, April 2021 Jacob Rosensköld, Magnus Sunesson

Contents

1 Introduction	9
1.1 Background	9
1.2 Problem formulation	12
1.3 Purpose	12
1.4 Related work	13
1.5 Workload distribution	13
2 Theory	15
2.1 Interaction Design & Cognition	15
2.2 Visualization	22
2.3 Graphs	25
2.4 Optimization problems	27
2.5 Development process	30
3 Method & Process	35
3.1 The work process	35

3.2	Tools and external libraries	38
3.3	Prototyping	41
3.4	Implementation	47
4	Final design	59
4.1	The graph	59
4.2	The toolbar	60
4.3	Final test	65
5	Discussion	67
5.1	Method and process	67
5.2	Development	68
5.3	User centered design	68
5.4	Final design	69
5.5	Future development	73
6	Conclusion	75
6.1	The research questions	75
6.2	The goals	76
7	References	77
8	Appendix A	81
8.1	Project plan and outcome	81
9	Appendix B	83
9.1	Design questions to Andrew	83
10	Appendix C	85
10.1	Mid-fi prototype, iteration 1	85

10.2 Mid-fi prototype, iteration 2	87
10.3 Final result	88

Chapter 1

Introduction

This chapter describes the background of the thesis and its purpose. An introduction to the problem being investigated is given, and a set of goals, as well as delimitations, are defined.

1.1 Background

Visualization is a fundamental part of humanity. Dating back as far as the stone age (Opila, 2019) cave paintings, different visualizations have helped humans transfer their ideas and thoughts to others throughout history. Today, we are constantly faced with different visualizations in our everyday life. Whether it be a visualization of tomorrow's weather or a spreadsheet in Excel, the usage of graphical elements is a powerful tool helping us convey and represent information. As computers have grown more powerful, allowing for more calculations and renderings to be done per time unit, data visualization has grown more powerful. We live in an information society where huge volumes of data are sent worldwide every day. As such, the need for proper visualization is undeniable.

This master's thesis explored visualization and design literature and used the knowledge gained to create a data visualization tool for the company Sinch. The visualization was done on a multi-layered data structure in the company and will help employees understand the relations between the different parts of their system. The tool that was developed is an interactive graph containing nodes and links, as well as necessary functionality to ease the understanding of the graph.

Sinch, being one of the largest companies in their market sector, has complex data systems. It can be hard to produce comprehensible engineering diagrams capturing all the relevant information at a sufficient level of detail. Typically today, these diagrams are drawn on an as-required basis, and have problems with becoming outdated and not linking together into an overarching perspective. To remedy this, a graphical representation can be used to visualize the architecture mapped into an object graph.

Sinch is currently using a prototype solution that visualizes its system architecture in graph form, as seen in figure [1.1](#) and [1.2](#). The system is divided into "Domains" (black boxes), "Services" (red boxes), and "Interfaces" (boxes with grey background). Service is a subset of Domain, and Interface is a subset of Service. Different Interfaces can be related to each other, both within the same Services, and to Interfaces located in other Services. When referring to subnodes in this thesis, what is meant is the nodes within nodes. Supernodes are nodes containing other nodes. Interfaces are subnodes to services, services are subnodes of domains. Domains are supernodes to services, services are supernodes to domains. Siblings are subnodes that have the same supernode parent. All domains are also siblings. While the current version works, there is room for improvement. We have been asked to create a tool that generates a graph and helps users interactively navigating this graph. It is also helpful to know how a system got to its present state, seeing what parts have been added, removed, and modified. Thus a function showing how the system has evolved, similar to a timeline, should also be added to the tool.

In this thesis, the data structure Sinch has provided is referred to as "multi-layered", meaning there are layers of nodes and subnodes as seen in figure [1.1](#) and [1.2](#).

1.1.1 Sinch

Sinch, formerly CLX Communications, is a Swedish company and a global leader in cloud communication with over 2000 employees and 145 billion API engagements per year. Sinch's product portfolio consists of APIs for messaging, calling, log-in verification and more. Using Sinch's solutions, companies can reach out to their customers on a global scale and, for example, remind them of appointments or keep them updated on their order's delivery status.

1.1.2 End-users

The ones that will use the tool are programmers and systems architects that want an overview of how the system is structured. Our supervisors from Sinch represent those end-users, and when referring to end-users in this thesis, those supervisors are what is meant.

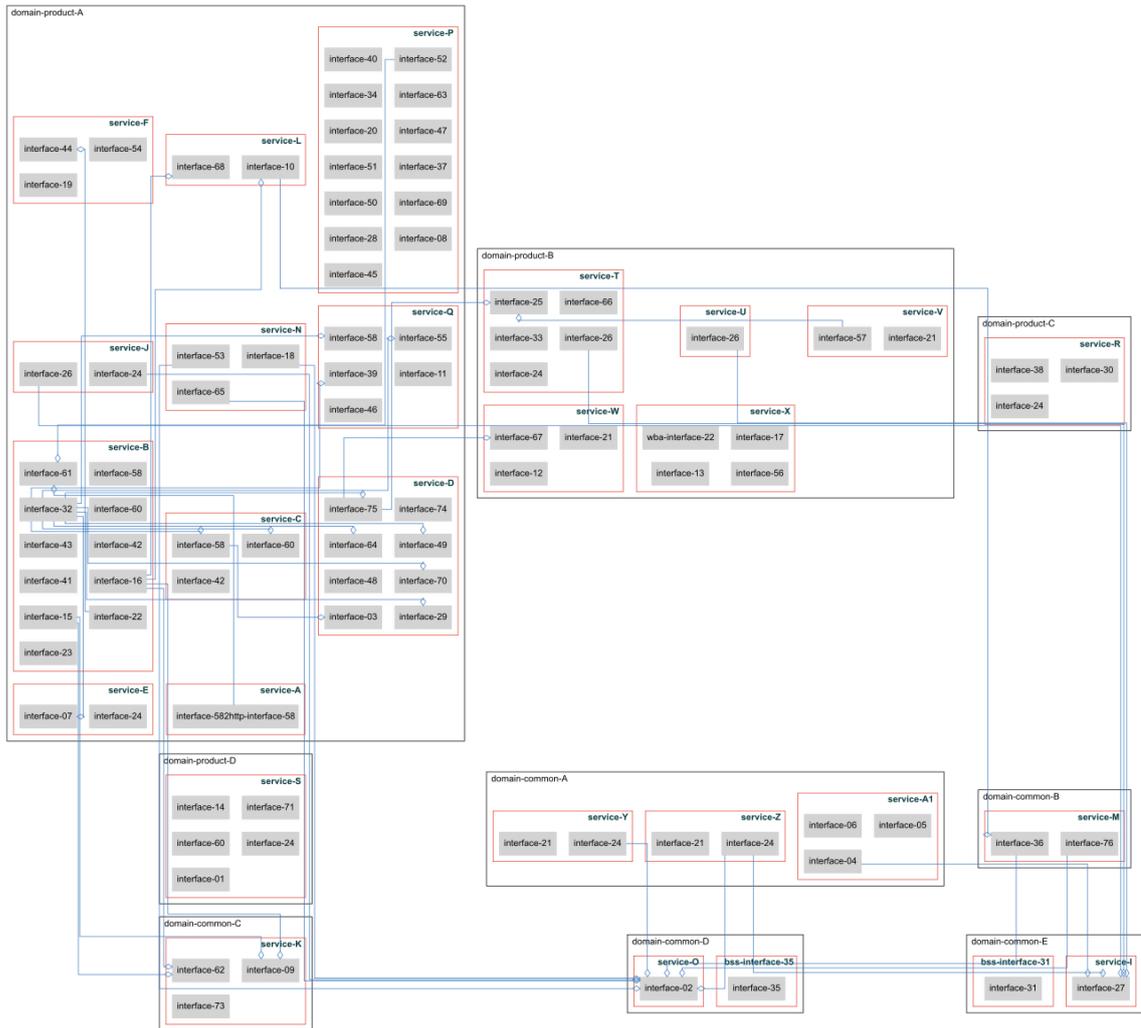


Figure 1.1: Sinch current prototype.

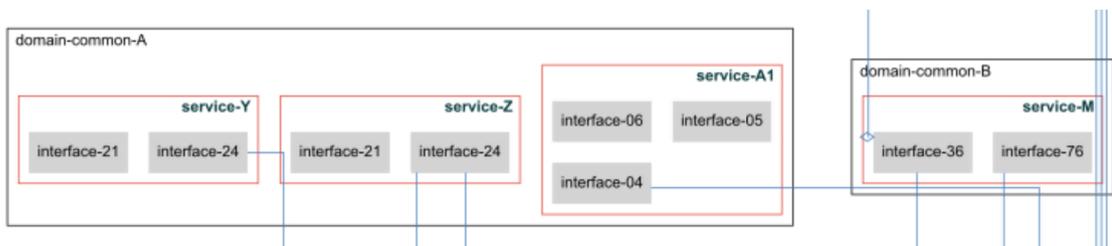


Figure 1.2: Sinch current prototype, zoomed in.

1.2 Problem formulation

Sinch has already built a prototype solution that describes the logical architecture of how the company's different applications relate to each other (figure 1.1). This has been done by transforming a directed acyclic graph (DAG) in JSON format to an SVG graph using Graphviz, a library used for graph visualization. The main problem with this solution is that it does not scale with the complexity of the DAG. The elements' placement is not optimized, leading to long links and several link crossings. The prototype produces a static SVG visualizing all elements and their relations, not considering that the user, in many cases, only will be interested in smaller subsets of the data. There is also a need to extend the prototype solution to describe the organization's common data model, which sites contain which applications, and the security architecture within the same graph.

1.3 Purpose

The purpose of this thesis is to propose a design for visualization of multi-layered data structures, as well as implementing the design using appropriate methods.

We aim to answer the following research questions in this thesis:

- How to visualize a complex system architecture, without losing too much information?
- Which factors will affect a users comprehension of a node networks?
- How can an application that visualizes the data structure be implemented?

We also have the following bullet points that will serve as goals of this thesis:

- Based on design and interaction principles, construct a prototype for effectively visualizing multi-layered data with relations. It should be clear which nodes are in the same group and which nodes are related to each other. This goal includes:
 - Use an appropriate design that gives an overview of the whole graph while making it easy for the user to focus on specific parts.
 - Hiding complexity in the graph and allowing the user to choose what data is of interest and hence, where complexity should be increased.
- Implement a tool based on the prototype. The implementation should work for generic data, meaning that evaluating and selecting suitable graph drawing algorithms must be done.

1.4 Related work

Visualization with node networks is a complex subject with many different aspects to keep in mind. Some literature that is of great importance to this thesis is the works of Purchase et al. (1996), Kobourov (2012), Kobourov et al. (2014), Shneiderman & Aris (2006) and Ware et al. (2002) due to their comprehensive work and frameworks regarding how to approach node networks visually.

It was decided to do the visualization in the 2D plane. The original prototype was in 2D and a transition to 3D was not something we considered. While a 3D solution would have been possible 2D was what we focused our literature study on. 3D is useful in that the designs can be closer to real-life metaphors (Teyser & Campo, 2009). Teyser & Campo also present many different ways to design 3D visualizations, some of which are related to hierarchical network structures similar to our multi-layered data. Those could have been alternative solutions to the 2D one presented in this thesis.

1.5 Workload distribution

The workload has been evenly shared throughout the whole project.

Chapter 2

Theory

In this chapter, the theory on which this thesis was based is presented. Some of the theoretical concepts are not used directly but are still included to give a deeper understanding of how the user's emotions and thought process may affect design choices.

2.1 Interaction Design & Cognition

Interaction design and cognition are two important disciplines to have in mind when designing a system to be used by humans. Using knowledge from those two fields will likely make the system easier to use and less prone to lead to frustration in the end-user. This chapter will go through some important concepts of those disciplines.

2.1.1 Background

Interaction design often refers to the process of designing products with an emphasis on how end-users interact with them. Preece et al. (2015, ch. 1.3) interaction design is described as "Designing interactive products to support the way people communicate and interact in their everyday and working lives" and "Put another way, it is about creating user experiences that enhance and augment the way people work, communicate, and interact".

In other words, the way user's interact, or would want to be able to interact, with a product

gives the designing process direction.

Cognition is a term that, put simply, refers to the conscious mental processes in our brains (Cambridge Dictionary). It is a broad and complex field. When we speak about cognition in this thesis, we refer to how the brain functions and processes information. Specifically how it relates to design choices, how you can use it to your advantage, as well as what to avoid to make products more useable.

2.1.2 Norman's principles

When designing a product, it is critical to consider who the users will be and understanding how the human brain works - why the users may do one action or another and how the user will feel about different designs. Of course, this may vary from person to person, but there are some characteristics that most people share that must be considered in the design process.

Norman (2013, pp. 40-44) describes seven steps a person could be said to go through before taking action, both consciously and subconsciously:

1. Set a goal: the user determines what they want to accomplish by using the product.
2. Planning: the user considers all the available options they can think of that could be used to achieve their goal.
3. Specification of action sequence: the user specifies which actions they will use and how they will execute them.
4. Execution of the actions: the user executes the actions.
5. Observation: the user observes the outcome of its actions.
6. Interpretation: the user interprets the outcome.
7. Comparison: the user compares the outcome with the goal set in point 1.

From these seven steps of action, Norman (2013, pp. 72-73) derives seven fundamental principles of design that every design should fulfill:

- **Discoverability** is one of the most critical design principles. It is about how well the user can detect what actions are possible to do at any given state of the design. Without proper discoverability, the user might not detect all available actions, leading to either taking an action that will not be the optimal one or not discovering any relevant action. Relevant actions for every given state in a design should be visible and easy to discover.
- **Feedback** is either positive or negative information telling the user what has happened after the execution of an action. This information is crucial to make the user feel in control. Without it, the user will not know if they have correctly used the product or if

something did not go as planned and is likely to redo the same action hoping for better results. Lack of feedback leads to frustration and a bad experience with the product. Overuse of feedback leads to confusion and could even be worse than too little of it. Feedback must always be direct. A good guideline is no longer than 100 ms. With proper feedback, the results of an action and the design's new state will always be easy to determine.

- **Conceptual model** helps the user understand how a product works by using concepts that the user will feel is natural and familiar from other contexts. For instance, it could be the folders on a computer that helps the user creating a mental model of how the files are organized. A good design should provide everything necessary for the user to create a relevant conceptual model.
- **Affordances** are the relationship between a design's properties and the user's capabilities of using it. If an affordance exists depends on how the user thinks that they can use the design. For example, the round doorknob on a door will, for most people, have the affordance to be turned and not pushed. The opposite of affordances is anti-affordances, which are things the user cannot do. A design should have all the necessary affordances so that the relevant actions are possible.
- **Signifiers** indicate to the user what actions are possible and where the user can interact with the design. Signifiers can be a mark, a text label, a sound, or anything else that will indicate the design's affordances. Signifiers should be used where the other principles may not be enough for the user to understand what actions are possible and the effect of using them.
- **Mappings** are about the relationship between controls and actions. With good and natural mapping, the user can quickly figure out an action's outcome without reading any instructions. Natural mapping means that the controls' design and placement correspond to the action's behavior and outcome. For example, in a slide show presentation, a natural mapping would be to place an arrow pointing to the right that moves the presentation to the next slide and an arrow pointing to the left that moves the presentation to the previous slide. What is natural is culturally dependent. For some, it could feel more natural to use the opposite order of the arrows in the example with the slide show presentation (right gives the previous slide, left gives the next slide).
- **Constraints** in designs prevent the user from doing things they should not do in a given state of the product. For instance, in an online form, a constraint could be an inactivated button, preventing the user from moving to the next step before filling in all required details. Using constraints narrows the set of possible actions to only the relevant, decreasing confusion and helps the user to execute appropriate actions.

2.1.3 The importance of aesthetics

Norman (2005), discusses the importance of appealing aesthetics in products. The basic premise here is that emotions are a big part of the human psyche, and can not be ignored

when designing products. Emotions have a big impact on our decision-making and affect how well a product will work.

Referring to Alice Isen (professor of psychology and marketing), he explains how feeling good will help with creative thought and problem-solving abilities. This is in stark contrast to when a person feels anxious. When anxious, tunnel vision and narrow thinking can occur. The argument is then made that things that look attractive make us feel better. This relates to design directly. When in the state of feeling good, users will have an easier time exploring alternative solutions for reaching the desired result when using a product, if they did not achieve it on the first try. Compare this to a more anxious emotional state, where it can be quite hard to solve unknown problems, such as finding functionality in a system. When in a bad state of mind, users tend to focus more on the details of what is not working, rather than looking for new solutions. The user may get stuck in a loop where it gets very hard to find the correct solution for its task due to tunnel vision, trying the same approach repeatedly. This will lower the system's efficiency. A user that is more relaxed and content will not have the same problem. The user in this state of mind will be more likely to try completely different approaches than those that did not work, exploring the system and trying new solutions. Norman further makes the case that a product that is fun to work with will render the user more likely to ignore negative aspects of it, focusing on the good.

In conclusion, emotions impact how well a user will interact with a product. A user's emotions can be manipulated through the use of proper design, and one should approach the design in different ways depending on how the product will be used. A product's aesthetic appeal thus can be said to make things work better or worse.

2.1.4 Ways to interact

There are several ways to interact with a system. Preece et al. (2015, ch. 2.5) mention four classifications:

- **Instructing** - The user makes the system to act a certain way by giving it instructions. By using control keys, selecting, clicking objects in a Graphical User Interface (GUI), the user can change what appears on the screen.
- **Conversing** - The user converses with the system in dialog form.
- **Manipulating** - The user manipulates objects in the system. Can be both virtual and physical.
- **Exploring** - The user explores a virtual environment to gain insight of it.

Manipulating is particularly interesting for our thesis and will be explored further here. Shneiderman (1983) came up with a framework called "direct manipulation". In this framework one should make use of real-life metaphors when designing a GUI, as it enables the users to easier gain knowledge on how to interact properly with it. He mentions four principles that characterize direct manipulation:

- Continuous representation of the object of interest.
- Physical actions (movement and selection by mouse, joystick, touch screen, etc.) or labeled button presses instead of complex syntax.
- Rapid, incremental, reversible operations whose impact on the object of interest is immediately visible.
- Layered or spiral approach to learning that permits usage with minimal knowledge. Novices can learn a modest and useful set of commands, which they can exercise till they become an "expert" at level I of the system. After obtaining reinforcing feedback from a successful operation, users can gracefully expand their knowledge of features and gain fluency.

Shneiderman (1983) also wrote about several benefits of using those principles:

- Novices can learn basic functionality quickly, usually through a demonstration by a more experienced user.
- Experts can work extremely rapidly to carry out a wide range of tasks, even defining new functions and features.
- Knowledgeable intermittent users can retain operational concepts.
- Users can immediately see if their actions are furthering their goals, and if not, they can simply change the direction of their activity.
- Users experience less anxiety because the system is comprehensible and because actions are so easily reversible.
- Users gain confidence and mastery because they initiate an action, feel in control, and can predict system responses.

Spritzer & Freitas (2008) describes how the need for properly choosing interaction and navigation are of equal importance as that of choosing the right kind of visualization. They also describe that the relation between visualization, navigation, and interaction is somewhat symbiotic, as one's effectiveness is partly determined by how the others are implemented. As for actual examples of interaction, they mention the following five techniques:

- **Filtering** - The user gets to decide what is highlighted or shown in the visualization by defining a subset with some criteria. For example a user could highlight nodes in a dataset that matches a certain criteria.
- **Searching** - The user inputs a search criteria to a searching mechanism, which returns node(s) in the structure that matches the criteria. The entities that fulfill the criteria are focused on.

- **Selecting** - By selecting specific elements in the dataset, the user can gain more insight about it. After selection the element could be highlighted and more information about it could be displayed in the visualization. Selecting can also be used in relation to manipulation and moving of the elements.
- **Zooming** - With big input data, a graph-type visualization of the data can get quite big. When the graph is too big, the user will get hard to distinguish the fine details. To deal with this, zooming can be used. There are two kinds of zooming, semantic as well as geometric. Geometric zooming simply makes the graph bigger. In contrast, semantic zooming changes the content in the visualization. More details are shown when the user approaches a specific area. A challenge with this approach is determining what level of detail should be shown.
- **Incremental exploration and navigation** - Only parts of the system are displayed, and as the user navigates through the system, for example, by clicking on hyperlinks on a website, new parts of the system reveal themselves. Incremental exploration can be used for graphs as well, displaying detailed information about an element that is in focus, while surrounding elements display less. This can, for example, be done by using a "fisheye camera" view or other visual techniques.

Spritzer & Freitas also mentions "Scrolling" but does not discuss it further.

2.1.5 User Experience

The user experience (UX) of a product is important. UX includes more than usability, it is the user's full experience - ranging from emotions, utility, usability, total satisfaction, and is an important part of interaction design (Mejtoft et al, 2019).

The Nielsen Norman Group, founded by two highly recognized individuals when it comes to UX, define it as "the user's interaction with company, its services as well as its products" (Nielsen Norman Group, n.d.). Norman, in another quote, states this in relation to user experience: "No product is an island. A product is more than the product. It is a cohesive, integrated set of experiences. Think through all of the stages of a product or service – from initial intentions through final reflections, from first usage to help, service, and maintenance. Make them all work together seamlessly".

McCarthy & Wright (2007, pp. 79-104) outline what they refer to as "the four threads of experience". The four threads are:

- **The sensual.** What level of sensory engagement does the user experience? How absorbed are they in the interaction? An analogy is made with video games, where users sometimes can experience full immersion in the experience.
- **The emotional.** Emotions are somewhat dependant by circumstances, and how we experience those circumstances. A user can become enraged and frustrated with a computer system depending on how it behaves.

- **The compositional.** The narrative of an experience. Online shopping is used as an example. With good structure the user can feel guided in a safe coherent way, while bad structure can lead to many questions arising. "What is this about? What has happened? Where am I? How do these things go together? What will happen next? Does this make sense? What would happen if . . . ?"
- **The spatio-temporal.** This refers to how experiences change how we perceive time, as well as how we perceive the space around us. Time can speed up or slow down, a user may feel as if they are being trapped if an experience is very frustrating, both depending on how the experience affects the user.

McCarthy & Wright (2007) also mean these threads to not be a necessary element in design, but rather a tool or framework for the designer to use. By considering those four threads, as well as Norman's definitions, designers can think and reason about UX in a more clear and concise manner (Preece et al., 2015, ch. 1.4).

There is a term related to user experience called user onboarding (Chapin, 2018). It refers to a concept where a system introduces a first time user to the system, by guiding it through the different functionalities.

2.1.6 Cognition

Knowing how the brain functions is beneficial when creating a system. Preece et al. (2015, ch. 3.2) describe how there are several modes of cognition. They mention thinking, remembering, learning, daydreaming, decision making, seeing, reading, writing, and talking. Preece et al. goes on to list different processes that can describe cognition. They go onto describe what implications those processes have on how one should chose to design a product or system. Preece et al. also refers to Norman (1994) , who classifies two kinds of cognition, experiential and reflective. Experiential is what happens without conscious thought, while reflective is a conscious thinking process.

- **Attention** is what is focused on. Can be both visual and auditory. If necessary, give information extra visibility when it needs to be done as part of a task. This can be done by using animations, colors, ordering of items and other similar techniques. Too much different information at a time makes it harder to notice the part of the system that needs your attention; If you use color to highlight the information that needs attention, it is a good idea to not have a great amount of other colors. Search engines and form fill-ins should have simple interfaces as that helps usability.
- **Perception** is how information from the external world becomes an experience. Sensory organs such as eyes, ears, fingers send signals that is experienced as sight, sound and touch. Graphical representations should be easy to immediately understand for users. Borders and spaces between elements are efficient for separating information and makes it easier to find the desired info. Text should be readable and easy to distinguish from its background.

- **Memory** is the process of recalling information received at an earlier point in time. Allows recognition and remembering of systems, environments, processes etc. A users memory should not be overloaded with complex information about how to perform a task. Recognition is more efficient than remembering, thus one should be consistent in placement of menus, use of icons and such. Users should be able to encode information to regain access of it easily in the future. Categories, tagging, colors and similar techniques are useful for this.
- **Reading, speaking and listening** - all three are language processing mediums. Whichever of those mediums is used, a sentence remains the same. Written language has a permanence that auditory language usually does not have. Text can quickly be scanned in a way sound can not. Speech-based interfaces and menus should be avoided as they usually are harder to follow than the alternatives. It can be a good idea to enable users to enlarge text, as some people have poor eyesight.
- **Problem solving, decision making, planning and reasoning** makes use of reflective cognition to go through what course of action to take, weighing options against each other and consequences. Using artefacts in the external world is common, such as pen and paper. A system should make it possible for users to access detailed information that usually is hidden, if the need for deeper understanding arises. Simple functions that are easy to remember should be used in computational aids, to improve the users decision making and planning.
- **Learning**, "the acquisition of knowledge or skills through study, experience, or being taught" (Lexico). For many, it is more efficient to learn by doing, than to follow a manual. Thus, interfaces should encourage users to do this. Allow them to make mistakes and undo actions, guide first time users towards the correct actions (Preece et al., 2015, ch. 3.2).

A system that the users interacts with to help them with cognitive tasks is part of something Hutchin (2000) called distributed cognition. He describes distributed cognition, explaining that oftentimes thinking is done in conjunction with something in the material world, be it pencil and paper, socially distributing your thoughts with other individuals or reading a text, which can relay the cognitive results of thinkers ages ago. Computer systems are very much a part of distributed cognition, with digital information distribution and intellectual support tools being more and more integrated to our everyday lives.

2.2 Visualization

Visualization could be described as the representation of information or ideas in a visual format. Visual formats include, but are not limited to, pictures, videos, sculptures or other arrangements such as art installations. We will focus on the two dimensional plane of pictures and images.

2.2.1 Background

Some systems are quite complex, with many entity relations to keep track of, subsets of data and such. Such data can be difficult to grasp for a user if it can only be seen in the visual format of "text". Using different visualization formats and techniques can help improve the user's understanding. What the correct visualization is depends on the nature of what is to be understood. The process of choosing the correct one is called visual mapping (Mazza, 2009, pp. 20).

2.2.2 Categorizing data using colors and contrast

Using colors is an effective way to categorize data when there is a small set of categories. The colors must be easily distinguished. The number of colors that can be used without significantly impairing the perception depends on the symbols' size representing the data. For small symbols, a maximum of ten colors is a useful guideline (Ware, 2013, pp. 124).

2.2.3 Representing relationships

Using lines Using lines to show relations is one of the fundamental gestalt principles and is a great way to show relationships. The lines should be continuous and have no sharp angles to make it easier to follow. Using different designs on the lines can represent different relationships, for example, representing current relations with solid lines and representing previous relations with dashed lines. By using different line weights, the strength of a relationship can be visualized (Ware, 2013, pp. 225).

Using proximity and similarity Placing data points close to each other will make them perceived as related, and having different densities of the points in the different groups will increase the effect. Having similar characteristics, such as color and shape, for related points will also make them perceived as being related (Ware, 2013, pp. 181).

Using regions If the effect of using proximity is not enough, using closed contours, forming regions, is a good option. Placing data points inside the same region will be more strongly perceived as related to each other than only using proximity. If the regions have complex forms or overlap, using colors and textures will make the regions more easily identified (Ware, 2013, pp. 186).

2.2.4 Direct attention to specific data

It is often useful to direct the user's attention to a specific point in the visualization, and there are several ways of doing so. There are cognitive aspects that are specifically related to visualization. Ware (2013, pp. 152-162) describes the phenomenon of pre-attentive processing. The phenomenon got its name from early researchers, who believed the brain processed certain information before giving it actual conscious attention. In figure 2.1 pre-attentive processing is demonstrated. By lowering the contrast/opacity in the right hand side of the picture, the task of finding all 3s is made easier.

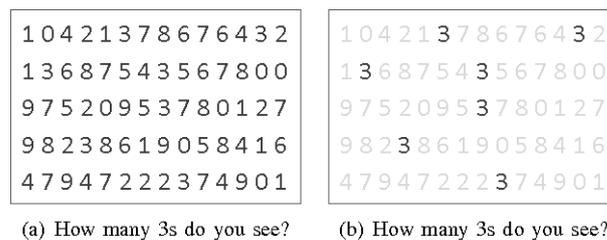


Figure 2.1: Illustration of a contrast based pre-attentive property.

Four categories are used to classify different pre-attentive properties; Form, Color, Motion and Spatial Position.

Some other examples of effective methods for directing the user's attention are:

- Changing the color of a specific point or area, using a color that clearly distinguishes it from the rest of the visualization.
- Changing the size, making a point or area larger than the surrounding.
- Using motion or blinking.

How well a method will work mainly depends on two factors; how much it will make a specific point differ from the other data points and how much the other data points differ from each other. If there are many points with different colors, it will still be hard to find a specific point even though its color is unique. If there are many points with different sizes, having one point larger than all the others may not have sufficient effect. To increase the effect and make a point stand out more, several characteristics, such as size and color, can be modified.

2.2.5 Chartjunk and design clutter

A variety of visual elements can be necessary for visualization of complex relations or systems. While creativity can be useful when creating a design, it is also important to not go overboard with visual elements. Tufte (2001, pp. 107) speaks about *chartjunk*, a term referring to unnecessary elements that are added to charts to decorate them, rather than to give

the user more information. If an element does not tell the user anything new, nor helps it navigate or understand the data, one should consider if the element is really necessary for the design. While it can make the data look more appealing, decorations solely for the sake of decor should be used sparsely.

Rosenholtz et al. (2010) discuss a similar phenomenon, *display clutter*. Similar to Tufte's case of avoiding chartjunk, it is important not to fill your design with distracting elements. In contrast to chartjunk that refers mainly to decorative elements of design, display clutter rather refers to the amount of elements on screen at any given time. When there are too many element on screen, it can lead to a cluttered state. Many elements are not necessarily a bad thing however, Rosenholtz et al. argues that, per definition of the American Heritage College Dictionary, a large amount of objects should only be considered clutter when it leads to confusion. They also describe many ways to avoid and decrease clutter, some of which are filtering and zooming techniques that were described in earlier sections. Further, they go on to describe how extra elements on the screen, while not necessarily confusing, in many cases can decrease users performances. For example, if there are many slightly similar objects crammed together in a dense space, it will be hard to find a specific one out of the lot. There is also an effect on short term memory, as there is a limit to how many objects one can keep proper track of at a time.

2.3 Graphs

Graphs are useful for representing information. There are many types of graphs, such as mathematical graphs that represent how a savings account grow over time, or family trees that show the relationships between family members. In our thesis we will focus on graphs that represent node networks, a graph that shows the relations between elements.

2.3.1 Background

Graphs are a collection of nodes with connections, edges, between the nodes representing their relation. With well-designed graphs, it can become a lot easier to get an overview of a complex system than describing the system in basic text form for example. Visualizing small graphs is an easy task, but with an increasing number of nodes, the visualization becomes more complicated. Showing too many nodes at the same time will make the graph hard to read. Show too few nodes and possibly vital information could get excluded.

2.3.2 Visualization in graphs

Mazza (2009, pp. 64) lists four things to have in mind when designing a data visualization in graph form.

1. **Positioning of nodes.** The nature of the data represented, usually, is such that there is no spatial positioning that is natural. Meaning, there is no clear way to organize elements based on how they would be arranged in the natural world. As such, the designer needs to define a fitting criteria for this arrangement.
2. **Representation of the edges.** The edges between nodes need to be defined. What kind of relations are to be shown, how are they best shown?
3. **Dimensioning.** For very large datasets it is simply not possible to visually represent all nodes and relations in an efficient manner. If the data you work with is very large, this has to be considered and solved.
4. **Interaction with graphs.** It is possible to interact with the graphical representation due to modern software. This can help solving the problem of large datasets, as well as lower complexity of a system. There are many ways to interact with a graph, which are mentioned in other parts of this thesis.

The fourth point, regarding interaction, is also mentioned by Spritzer & Freitas (2008). They claim that interaction and navigation is vital for bigger graphs, as it is very hard to find the desired information without those aspects implemented. As graphs get more complex the more data they represent, the need to navigate the data arises. Too many nodes representing elements of information, and too many edges located in a small space can make it outright impossible to gather information from the visualization. This can be solved with efficient layouts, reducing the information displayed or with interactivity (Mazza, 2009, pp. 66-72).

2.3.3 Visualization in node networks

According to studies by Purchase et al. (1996), Purchase (1997) and Kobourov et al. (2014) minimizing the number of edge crossings is the most significant factor affecting comprehension. Kobourov et al. (2014) further states that the effect of edge crossings is more significant on smaller graphs than it is on larger ones. Minimizing the number of bends of the edges has a smaller effect than edge crossing, though not negligible (Purchase et al., 1996, Purchase, 1997). Another study by Ware et al. (2002) states that the degree of bendiness is an essential metric for finding paths between nodes. In some cases, it can be better to have more crossings if it results in lesser bendiness. Increasing the ability to find paths can also be achieved by merging several edges that lead to the same destination (Grant, 2018, pp. 178).

Shneiderman et al. (2006) comes to the conclusion that the best network visualizations consist of 10 to 50 nodes and 20 to 100 links. Numbers above those will increase the number of links that cross each other, a phenomenon which lowers comprehensibility. Too high a number of nodes and links will also make it hard for the user to actually comprehend the relationships. This is part of why filtering techniques, such as reducing the number of nodes on screen, are vital to network visualization.

When drawing networks, Shneiderman et al. (2006) suggests following a set of challenges, in the order they are presented.

1. **Basic networks** with nodes and links. Nodes are unlabeled points and links are undirected.
2. **Node labels** (e.g. article title, book author, or animal name)
3. **Link labels** (e.g. strength of connection, type of link (active/inactive, car/train/boat/plane)).
4. **Directed networks** (links go from source to destination, such as from citing to cited article in citation networks or from predator to prey in food webs).
5. **Node attributes** that allow meaningful grouping (spatial layout), coloring (continuous or discrete), or sizing (continuous or discrete) of nodes:
 - categorical (e.g. journals/conferences/books/websites or mammals/reptiles/birds/fish/insects)
 - ordinal (e.g. winter/spring/summer/fall or small/ medium/large)
 - numerical (integer or real) (e.g. age or weight)
6. **Link attributes** that allow coloring (continuous or discrete) or thickness coding (thin or thick):
 - categorical (e.g. car/train/boat/plane)
 - ordinal (e.g. weak/normal/strong)
 - numerical (integer or real), (e.g. probability, length, time to traverse, strength)

2.4 Optimization problems

Optimization is defined as "*the act of making something [...] as fully perfect, functional or effective as possible*" (The Merriam-Webster.Com Dictionary, 2021). For many problems, optimization and the problem itself can be described in mathematical terms. An optimization problem is when the best, whatever that term may entail, solution to a problem by using some sort of mathematical approach are sometimes called *optimization problems*.

2.4.1 Force-directed graphs

For a computer to generate graphs with generic input data, algorithms must be used to compute the nodes' positions. With *Force-directed algorithms*, optimal node positions are computed by simulating physical forces between them. The physical forces can be, for example, gravity or electrostatic charge (repulsion). Hu (2012) argues that when the algorithm involves continuous calculations of force application on vertices until the system reaches equilibrium, a balance. Graphs generated by using force-directed algorithms often look aesthetically pleasing and produce layouts with few edge crossings. They may not be the best solution for graphs with a very large amount of elements (Kobourov & Pupyrev, 2014).

2.4.2 Label placement

Labels are a common part of visualizations. They help users understand what a data element is referring to. In a geographical map, the name of a city, next to the element representing the city, would be a label. Label placement refers to the process in which labels get their positioning within a map, graph or other visual representation. Without proper algorithms, there is a risk that the labels get placed in a way that lowers the visualizations usability. Based on Imhof and Yeoli's research, Wang (2013) presents three basic rules for label placement:

- Spatial overlap: There should be no overlap between labels and other important elements.
- Unambiguity: It should be clear which specific graphical element a specific label is related to.
- Legibility: Labels should be easily readable.

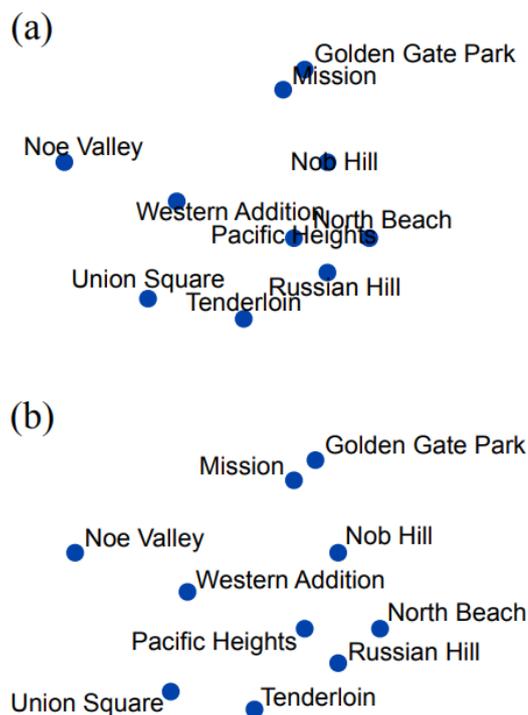


Figure 2.2: Two kinds of label placement.

In figure 2.2 the difference between following or not following these principles can be seen. In figure 2.2(a), the principles are not followed and there is a lot of overlapping of both labels and data points, and since the placement is done without following a standard there is ambiguity on which label relates to which data point. In figure 2.2(b) the principles are followed, leading to a more readable graph. Imhof also suggested three principles that would make the placement more stylistic:

- Placing labels on the right side of a data point is preferred over the left.
- Placing labels on top (over) the data point is preferred to placement below.
- Placement close to the corresponding data point is preferred.

There are several different implementations of algorithms following these principles, each with different ways of optimizing the resulting label placement. A concept used in most of those is the use of an energy function, in which the energy can be used to understand how efficient an algorithm was or was not. Energy is based on:

$$E_i = f(A_{ll}, A_{lp}, L_{lp}, \theta)$$

Where A_{ll} stands for area of label-label overlaps, A_{lp} refers to the area of data point - label overlaps, L_{lp} is the distance between a label and its point, and θ is the relative orientation for a label to the point.

To calculate the energy for a label the following formula is used:

$$E_i = \sum_{j \neq i}^N A_{ll} W_{ll} + \sum_{j \neq i}^N A_{lp} W_{lp} + A_{lp} W_{lp} + \theta W_{\theta}$$

where the W terms are weight constants, which are used to signify the importance of each rule. For each i , all we sum over all other labels and points. This gives the total overlapping area. High numbers of overlap results in higher energy. For the total energy, every label is summed for:

$$E_{tot} = \sum_{j \neq i}^N E_i$$

One is the Greedy Algorithm. In the Greedy algorithm, labels are placed serially, in a way that minimizes the current energy, based on the other labels that have already been placed. It does not take into account what future placements may mean for the total energy. Thus it is not very well suited for problems where taking the future into account is beneficial. The algorithm is usually very fast, but due to these problems it often generates placements that have room for improvement.

A second one is called Gradient descent. It is an algorithm that optimizes a specific configuration of label placements by doing local changes to labels. It randomly makes perturbations to the configuration and keeps track of every perturbation's ΔE . The one with the ΔE that makes the most difference to the original one is chosen as the new configuration. A problem with Gradient descent is that it only makes local changes, it is a local optimization technique. Its results are heavily dependent on the initial placement of labels.

A third one is called Simulated annealing. This is a global algorithm which will not depend as heavily on initial placements as Gradient descent. Similar to the Gradient descent it tests random changes to the configuration and updates the current one when acceptable. The difference is that it does it globally and thus can find solutions with less regard for initial placement.

2.5 Development process

There are many ways to develop a product. Frameworks for how the process could be outlined can make the work process more efficient. In this chapter some common aspects of developments will be presented.

2.5.1 Prototyping

A prototype can be just about anything, as long as it gives a sense of the final product. How similar the prototype should be to the final product depends on what stage the development process is. At an early stage, the prototype might be drawings on paper. At a later stage, interactable software models or physical devices might be more appropriate. The purpose of prototyping is to make sure that the designers have correctly interpreted the requirements before they start working on the final product (Preece et al., 2015, ch. 11.2.2). Arnowitz et al. (2006, ch. 2) presents a method for efficient software prototyping, consisting of four phases, described below.

1. The planning phase

To start the prototyping, the designers should collect requirements about the product related to different aspects of the product, such as technical, usability, functional, and business. The list of requirements must not initially be complete and will gradually be updated. All requirements are viewed as assumptions until they are validated by the user. Prototyping is an iterative process, and for each iteration, the requirements will either be validated, modified, or removed. The requirements should be prioritized as well as put in context to various actions. The planning phase also includes determining how the prototype should work the level of prototype fidelity. Usually, it is best to begin with low fidelity and gradually increase the prototypes' level of details as the project evolves. With too high fidelity at an early stage, much time will be spent designing features that later might be discarded.

2. The specification phase

At this stage, the prototype's main characteristics are established, such as who will view the prototype, how long the prototype should be used before making an upgrade, and at what stage the prototyping should take place. This stage also includes specifying what methods and tools the prototyping will use.

3. The design phase

The prototype is designed according to the established requirements., using conventional design practices, with the selected methods and tools.

4. The results phase

The results phase is about evaluating and testing the prototype. This is done by both the designers and a suitable audience.

2.5.1.1 Prototype fidelity

Prototype fidelity is how detailed and similar a prototype is to the final product. Both low fidelity and high fidelity prototyping are described below. However, it is best practice to gradually increase fidelity from low to high during the prototyping process, not reaching high fidelity too early (Arnowitz et al., 2006, ch. 5).

Low Fidelity Prototyping

As defined by Preece et al. (2015) the purpose of a low-fidelity (lo-fi) prototype is to investigate different concepts at an early stage. It should therefore be easy to modify and not include any complicated functions or design features. They do not have to be very similar to the final product. For instance, they could consist of drawings on a piece of paper instead of computer graphics. An advantage with lo-fi prototyping is that it can be done quickly.

Middle Fidelity Prototyping

Medium-fidelity (mid-fi) prototypes should be more detailed than low-fidelity and convey a more realistic picture of the final design while remaining easy to modify (Arnowitz et al., 2006, ch. 5). Software tools can be used to produce static images for explaining design choices and the behavior of different interactions.

High Fidelity Prototyping

As opposed to lo-fi prototyping, Preece et al. (2015) states that a high-fidelity (hi-fi) prototype must look and feel more like the final product. A hi-fi prototype should include enough functionality and design properties to enable interaction and testing. Before making a hi-fi prototype, design and usability requirements should have been validated through previous low and mid-fi iterations.

2.5.2 User centered design

During the prototyping process, it is vital to conduct user tests, where end-users get to try the current prototype. Letting users test the product during its development phase, rather than at the final product stage, lets the designer get valuable feedback on what is done. The end-users are the ones who know what needs the product is going to fulfill, and thus can give pointers on whether the designer is going in the right or wrong direction. An open discussion between the end-user and the designer can lead to new ideas as well as deeper

understanding of the requirements. How closely designers will work with end-users depend on many circumstances (Preece et al., 2015, ch. 9.2.2). The process of communicating with the users about their experience with the product, what requirements they would like fulfilled and iteratively letting them give feedback is often called evaluation. Preece et al. (2015, ch. 13.2) discusses four questions regarding evaluation - Why, What, Where and When.

Why - It can shed light on real problems or challenges with a product, enabling the designers to take a more correct approach rather than having to rely on their own opinions of what the product should be like.

What - Just about anything can be evaluated; aesthetics, workflow, functionality and so forth. Many analogies are made, one of them being "A software company may want to assess market reaction to its new homepage design."

Where - Oftentimes web accessibility and such elements are evaluated in a laboratory, but it depends on the element being evaluated. Sometimes a more natural setting, such as a child playing with a new toy at home is more appropriate. Those are called "in the wild"-studies.

When - The prototyping stage described earlier is part of the "When":s of evaluation. Evaluations performed during the design (after prototyping stage) are called formative evaluations, which help the designers to iteratively improve the product by using user feedback. When a finished product is evaluated it is called summative evaluation. Here the aim is to, if necessary, improve the final product in various ways. Features can be added or improved.

2.5.3 Agile software development

There are many ways to plan your work in software development. A common one is the waterfall model, where requirements and the work process is defined in quite great detail (Preece et al., 2015, ch. 12.1). In contrast to this, there is also the agile software development model. Here only basic requirements are defined in such a way that the developers can start the work process. Those requirements can be changed during the development and change is embraced (Manifesto for Agile Software Development, 2001). There are many different approaches in agile software development, such as Scrum, Extreme Programming and Kanban (Hughes & Cotterell, 2009, pp. 93). One thing most have in common is that they let the development happen iteratively (Agile Alliance, 2021).

Scrum can be described with these four elements (Scrum Guides, 2020):

1. A Product Owner orders the work for a complex problem into a Product Backlog.
2. The Scrum Team turns a selection of the work into an Increment of value during a Sprint.
3. The Scrum Team and its stakeholders inspect the results and adjust for the next sprint.
4. Repeat.

The Product Owner is the person who is accountable for making the value of the Scrum teams effort maximized, and in extension is also accountable for Product Backlog Management. The product owner is always one person rather than a team of several people. A Scrum Team is a smaller team of people who develop the product. A Product Backlog is a list of things that need improvement. The Scrum Team's work consists only of tasks from the Product Backlog. A Scrum Master is responsible for implementing Scrum practices into a project. That the Scrum framework is incomplete is not by accident, this is purposefully done so that the people using Scrum will fill in the details on themselves, based on their project's specific needs.

In Scrum, the iterative life cycle revolves around what is called "sprints". The sprint is a short time period in which the active development is done. Goals for each sprint are predetermined, and a sprint is fulfilled when the goals are reached. After a sprint is finished, its result is to be reviewed and evaluated during a Sprint Review. Here, which goals have been met and changes in the environment, such as increased complexity due to an implementation, can be discussed. The information gathered during a Sprint Review can then be used for deciding what the next step in the development should be. There is also a Sprint Retrospective to be done. How things went during the last sprint is reviewed, with focus on individuals, interactions, processes, tools and their definition of done. The Scrum Team evaluate the sprint and identify what can be improved. After the Sprint Retrospective is done, the sprint is over.

A tool often used in Scrum to increase its effectiveness, is a *task board* (Cohn, n.d.-a). In a task board the different tasks that already have been done, or are still to-be done, are visualized on some kind of board. The team can write stories about use cases. This card is then placed in the "Story" column. Each subsequent column are to be filled with cards related to the corresponding story for every row.

A demonstrative example is seen in figure 2.3.

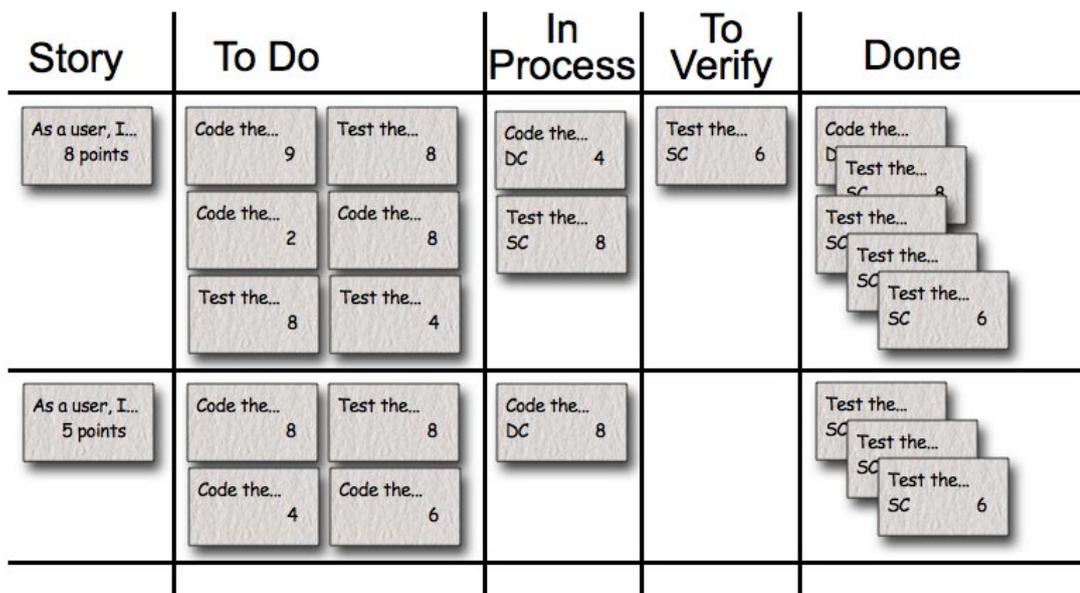


Figure 2.3: A basic Scrum Taskboard.

- In the Story column, stories about use cases are placed. These stories form rows with related to do cards, in process cards and so forth.
- In the To Do column, team members place cards with tasks that are to be done. This can be updated at anytime, if someone were to come up with a task during the sprint for example.
- In the In Process column, the cards that are being worked on at the moment are placed.
- In the Verify column cards that are related to testing are placed. If there has been a card for "Implement code A", there are likely some cards regarding testing this, such as "Test code A for use case X" and "Test code A for use case Y".
- In the Done column, cards with tasks that are finished are placed. They get removed when a sprint has ended, or earlier if there is too many cards.

For example, a story could be something such as "user wants to click button X to close a window". Then the "To Do" column could be filled with task cards such as "implement closing functionality for the window", "add design for button X", "test if button X's closing functionality works" and so forth. During the sprint team members can pick a card from the "To Do" and move it to the "In Process" or "To Verify" column. As the cards from "To Do" get handled, they will end up in the Done column. One of the advantages with using a Scrum Task Board is that it directly visualizes the sprint backlog, giving a quick graphic overview.

Chapter 3

Method & Process

This chapter describes the chosen methodologies and how they were used, based on the theory, to obtain the results. The chapter includes the work process, tools used for prototyping and development, requirements, prototyping and finally the implementation of the tool.

3.1 The work process

3.1.1 Introduction

To begin the work, the needs and wants of Sinch were clarified. A goal document was made that was approved of Sinch. After this, a literature study was done on fields such as data visualization, interaction design, user experience, graph design and more. Following the literature study, iterative prototyping with feedback was done, finally leading to implementation. See figure [3.1](#) for an overview of the complete work process. For the cyclic iterative process of prototyping and implementing, see figure [3.2](#).



Figure 3.1: General work process

We wanted to combine knowledge from the literature study with the continuous feedback received from end-users to develop. Due to Covid-19, user tests in person could not be per-

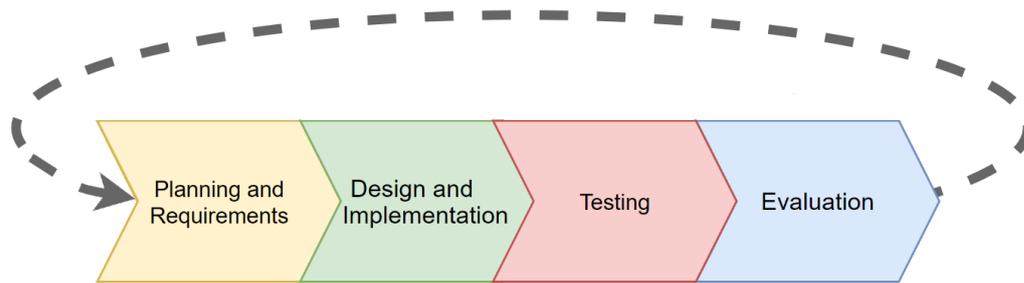


Figure 3.2: Iterative and incremental development.

formed, so instead showing and telling about the design was done online with screen sharing. While it does not follow the traditional practice of user studies, valuable feedback was received nonetheless.

The end-users would likely be quite technically proficient. It is also likely that they have some kind of knowledge about the data structure that is visualized. That affects the design in several ways. They can be expected to be able to handle interaction where the keyboard is combined with the mouse, such as holding down the "Shift"-button and at the same time left-clicking an element. When the end-user is a quite well defined target group one can also benefit greatly from involving them in the development process (section [2.5.2](#)). While they do not always know *how* they want the product to do a thing, they often know *what* that thing is. How we involved the users is discussed further in future sections.

3.1.2 Planning

To maintain flexibility in the process, we adopted an agile development approach based on Scrum. As we were only two persons, we needed to make some adjustments to the method. Instead of having a Scrum Master we cooperated to implement Scrum in the project. The main activities and tools were:

- Weekly meetings
 - Update and review
 - Goal planning
- Trello board with:
 - Stories
 - To do
 - Verify
 - Work in process

– Done

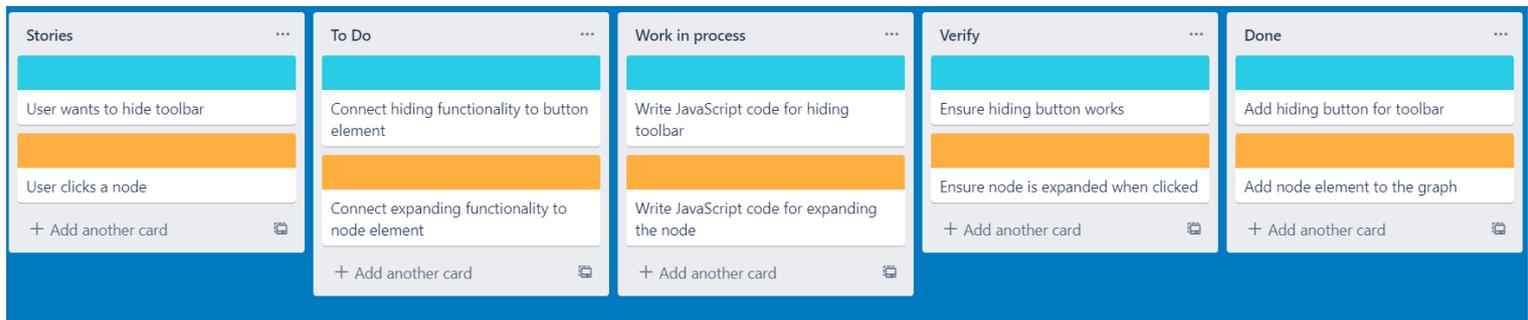


Figure 3.3: A Scrum Taskboard in Trello. Color coding is used to separate different stories.

The main advantage of weekly meetings the coming week got planned was adaptability to unexpected problems or changes. Also, reviewing the past week like this helped identifying problems.

For the Trello board, the main advantages was having a quick visual overview over the past, the present and the future of the project. This helped with insights on what paths to take. See figure [3.3](#) for an example of how it could look.

3.1.3 Requirements

When designing prototypes and implementing, different principles from the literature study such as Norman’s interaction principles, gestalt laws and visualization theory were followed. While some of them are non-quantifiable, having them in mind when prototyping and implementing is still beneficial. The following three components guided our design to some degree. **Hiding** information, to lessen the risk for design clutter (section [2.2.5](#)), and letting the user interact with the graph, which’s main focus would be **discoverability** (section [2.1.2](#)) and **manipulation** (section [2.1.2](#)) of the information. By having those interaction elements, and many other, we try to follow Spritzer & Freitas’s claim about the importance of interaction and navigation for big graphs (section [2.3.2](#)).

Based on the literature studied and the requests from the company, some minimum requirements for the tool was formulated. These are not very narrowly defined due to the agile development approach. Instead, they define a loose set of functionality, with several sub requirements as to how to implement them. The minimum requirements were:

1. The user should be able to get a good overview of the data by using the tool.
2. The user should have a high degree of control of how much information is shown at any given time.
3. The user should be able to switch between different data files.

4. Let the user undo actions.

Requirement 1 and 2 was mostly based on wanting somewhat concrete yet vague goals to work towards. Designing for a good overview and high degree of control could mean many things. Since it was uncertain which kind of visualization would be the best fit for the data structure yet, those open-ended formulations were chosen.

Requirement 3 was based on the problem formulation from the company combined with the knowledge from the literature study. Requirement 4 is based on the point about Learning (section 2.1.6). Letting users undo possible erroneous actions can increase willingness to explore and learn about the system.

There was also some desirable requirements. They were not of an as basal importance as the three above, since they do not relate directly to what is absolutely necessary for the product to work, but they are quite needed nonetheless. The product should look somewhat aesthetically pleasant, heightening its usability according to Norman's reasoning regarding aesthetics (section 2.1.3). While this is subjective there are still ways to achieve this on a general level. It should also implement principles for graph interaction discussed earlier in the report, such as zooming, filtering and highlighting, all to make the information processing smoother for the user.

3.2 Tools and external libraries

3.2.1 Figma

For mid-fi prototyping, the online tool Figma was used. It is a tool where you can make prototypes online collaboratively. It is somewhat similar to image editing programs, as you can choose from a preset of shapes and use drawing tools to create the visual elements of the prototype. Neither of us had any former experience in this tool. This was not a problem, as the tool was very user friendly. In our project, we had several different "pages". A page is as it sounds, a page with design elements. A useful feature was that you could duplicate pages easily, making it useful for making several different variations of the same design to find the best one. A page consists of elements, such as circles, lines, rectangles.

3.2.2 Graph drawing library and D3

Since drawing optimal network graphs, optimizing node placement and minimizing link crossings, is a complex problem, a decision was made to rely on an external library for the graph computations. A list of criteria was defined and used when comparing libraries:

- Performance of the graph drawing algorithms: the most crucial criterion and included how well the library performed to place nodes and minimize link crossings.
- Flexibility: how much the behavior of the algorithms was customizable without having to change the source code. Since the design consisted of multilayered nodes, with several partially or fully disjointed networks, having great flexibility in affecting the algorithms' behavior was necessary.
- Available documentation: an important criterion to understand and get the most out of the library.
- Others: available example code, popularity on Stack Overflow and other useful functionalities, such as visualization tools.

Several libraries were studied and rated after how well they met the criteria and the library that met the criteria the most was D3. D3 stands for Data-Driven Documents and is an open-source JavaScript library that includes powerful and customizable force-directed layout algorithms for computing node positions in networks. D3 includes, among others, the following forces:

- **forceCollide**: A circular force, ensuring that, for any two nodes, the distance between their center points is at least the sum of the nodes' specified force radius. For example, if node a has a force radius of r_a and node b has a force radius of r_b , node a and b will not get closer to each other than $r_a + r_b$, measured from their centers. See figure [3.4](#).
- **forceCenter**: A force that uniformly affects all nodes, pulling them towards the specified center point without changing the nodes' relative positions.
- **forceLink**: A force affecting nodes that are linked together. This force can be used both as an attracting and repulsing force, depending on desired link distance.
- **forceManyBody**: A force that works as either gravity or repulsion. This force, unlike **forceLink**, is applied to all the nodes in the simulation.
- **forceX/forceY**: These are positioning forces that push the nodes toward a specific position on the x or y-axis. See figure [3.5](#).

For computing nodes' positions in a network using D3's force-directed algorithms, a D3 simulation object is used. With the simulation object, one can define which forces to be used. Starting the simulation, it runs for a number of steps (default 300) and applies the forces on all nodes in every step. The forces can be set by changing their strength and the number of iterations that they should be applied on the nodes in each simulation step. It is also possible to define custom forces.

In addition to that, D3 also provided several useful visualization tools. It was well tested and had a lot of available examples, as well as well-written documentation.

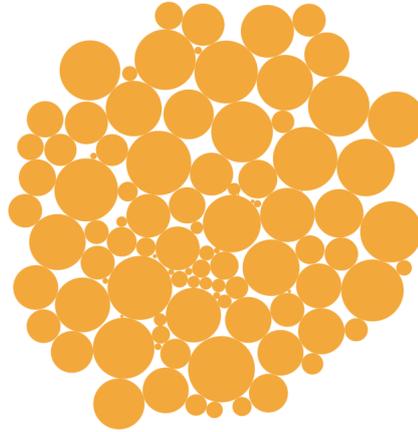


Figure 3.4: Force collide. Ensures distance between the nodes. Force radius for each node is set to the node's radius.

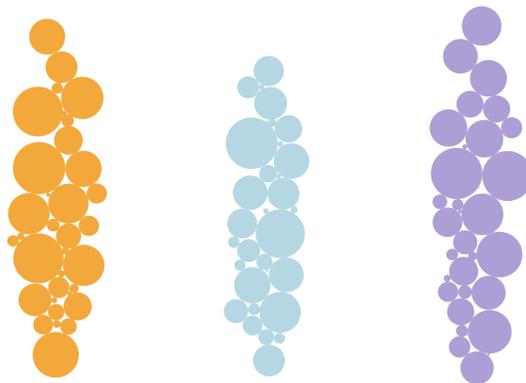


Figure 3.5: Position force X. Pushes the nodes toward a specific position on the x-axis. This example uses three position forces, one for each color, as well as force collide to avoid nodes overlapping.

3.2.3 Label placement library

For label placements, we have utilized a library called D3-Labeler. This library implements a simulated annealing algorithm for finding optimal label placements, as described in section [2.4.2](#)

3.3 Prototyping

This section describes the iterative process of developing the prototype that was later used during the implementation. Both lo-fi and mid-fi prototypes were made, following the phases described in section [2.5.1](#). One of our end-users, who is a systems architect, were particularly involved in the project. This end-user was involved during the process to ensure that the design evolved in the right direction section [2.5.2](#)

3.3.1 Lo-fi

The lo-fi prototyping was done in two iterations, using pen and paper. These prototypes aimed to explore overall layouts and did not include colors and other design details.

Iteration 1

Before creating a first prototype, an analysis of the current prototype by Sinch (figure [1.1](#)) was done. This was done by comparing the current prototype with the suggestions on what to consider during graph visualization as well as node network visualization (section [2.3.2](#) and section [2.3.3](#)). We found that the key problems with Sinch's prototype were:

- Arbitrary position of the nodes, leading to long edges and several edge crossings.
- No way of hiding data, that is, no way of decreasing the complexity of the graph.

From this, as well as requirements 1 and 2 from section [3.1.3](#), it was decided that the first lo-fi prototype should include the following properties:

- The nodes should be represented as round objects instead of rectangles. This makes it easier to position them such that link crossings are minimized, essential to make the links' path more evident (section [2.3.3](#)).
- Dependencies between nodes should be visualized using lines without any sharp angles, increasing the ability to identify the connected nodes (section [2.2.3](#) and section [2.3.3](#)).

- Since natural groupings can be seen in the data, these should be formed in a similar way as Sinch's prototype. That is, services belonging to the same domain should be grouped, and interfaces belonging to the same service should be grouped. The grouping should use both proximity and regions (section 2.2.3).
- Through interaction, the user should have the ability to choose how much of the data is visible at any time.

In figure 3.6 the results from iteration 1 is shown. These lo-fi prototypes were simplistic but served as a basis for the following iterations. In figure 3.6(a) the graph is shown with the lowest complexity, meaning that only domain nodes and their dependencies are shown. Complexity can be increased by changing the "level of complexity" on the right-hand side. When the user increases the complexity, more nodes become visible. In figure 3.6(b) the complexity level has been increased to show all services, that are *subnodes* of the domains. The links between the domains have now been redrawn, showing the dependencies between the services instead.

Iteration 2

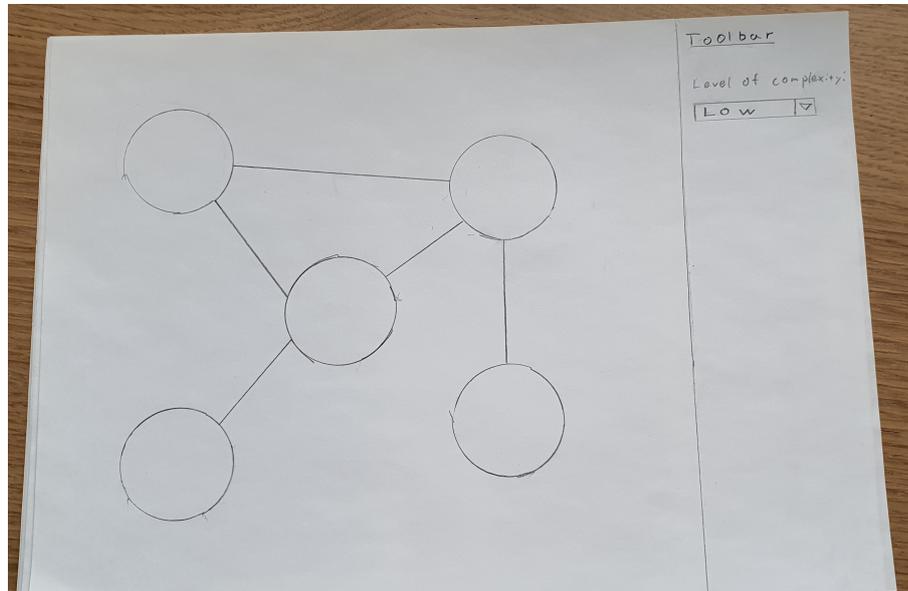
Further discussion with one of our end-users, the system architect from Sinch, and clarification of their needs, lead to the following prototype modifications:

- The user should be able to click on regions to get a more detailed view.
- It should be possible to select nodes, making them, as well as their links, highlighted.
- A detailed text description of the selected node should be visible.
- It should be possible to search for nodes.

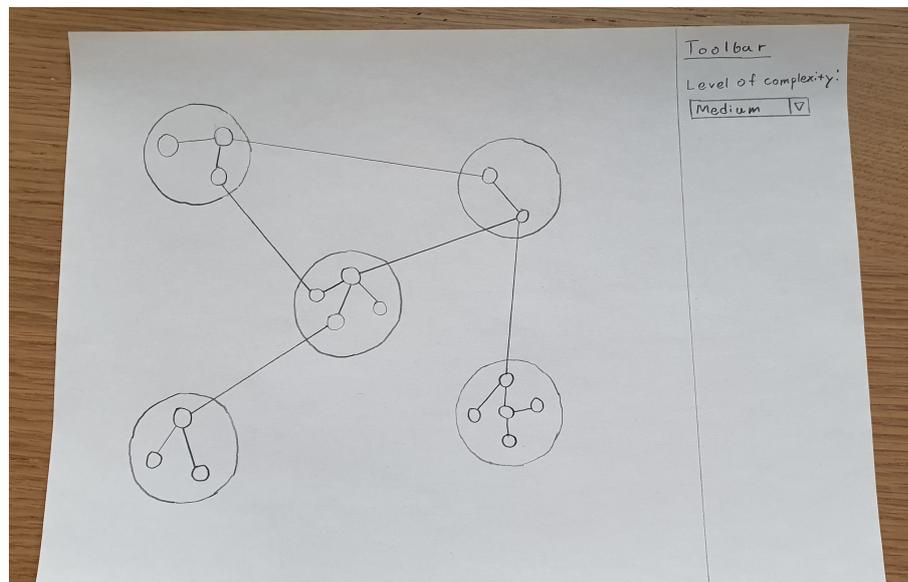
In figure 3.7(a), an overview of the graph is shown. Labels have been added to the nodes, and by selecting a node, both the node's border and the node's links are highlighted by making them thicker. An information box has been added, displaying a description of the selected node. In the toolbar on the right-hand side, the user can search for specific nodes. Searching for a node makes the node selected, as with node 58 in the figure. In figure 3.7(a), a zoomed-in view is displayed. This view is shown if the user clicks on a region, such as a domain. In this view, the user sees all the nodes in the region with their respective labels.

3.3.2 Mid-fi

The purpose of the mid-fi prototype iterations was to further develop the design from the lo-fi iterations. Figma (section 3.2.1) was used to create detailed prototypes, which was easier than the lo-fi prototypes for our end-user to understand. These prototypes enabled better understanding of the interactions and their potential drawbacks.

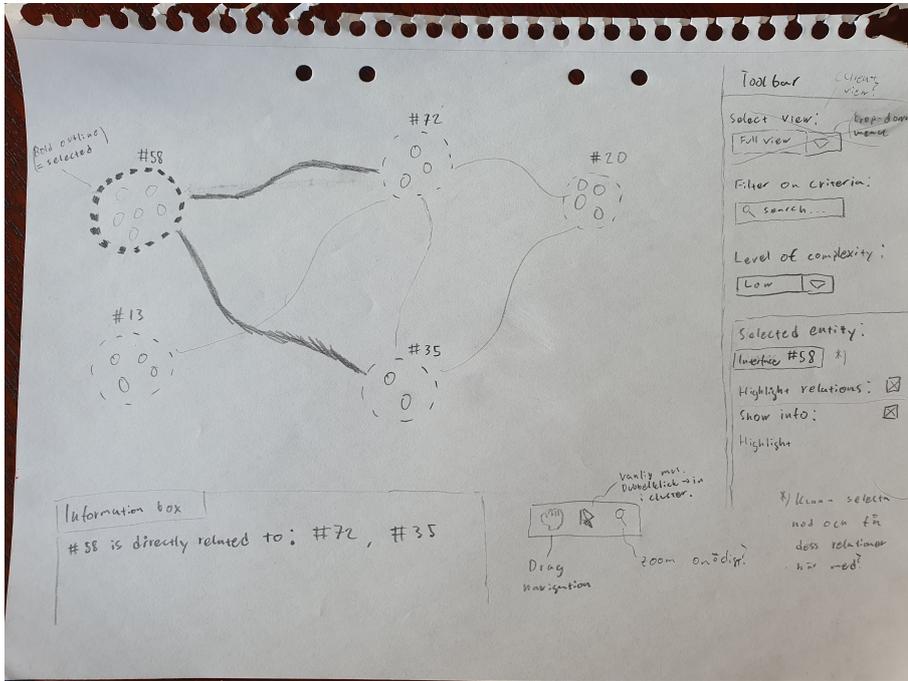


(a) Low complexity, only including the domain nodes.

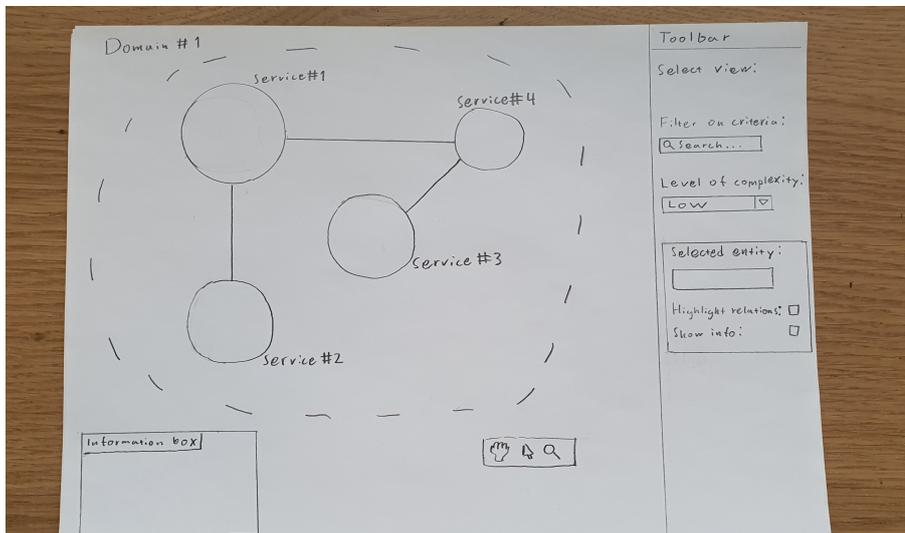


(b) The complexity has been increase, showing both domain and service nodes.

Figure 3.6: Lo-fi prototypes iteration 1. (a) Complexity is set to show only domain nodes and their dependencies. (b) Complexity has been changed to show both domains and services.



(a) Overview of the graph. Node 58 has been selected.



(b) A zoomed in view of a region.

Figure 3.7: Lo-fi prototypes iteration 2. (a) An overview of the graph. Node 58 has been selected, shown with a thick border and links in bold. (b) A zoomed-in view of a region.

The problem, visualizing the data structure given, we had to solve was quite complex and it took many iterations, internal discussions as well as feedback from our end-users to get to the design we finally decided on. While the core idea behind the final solution we settled on was similar to one of the earlier ones, the design of how the graph would behave was very different.

First iteration

The aim of the first mid-fi iteration was to convert the lo-fi prototype into a more detailed version in Figma. In figure 3.8 two images from the prototype are shown. See appendix C for more. Color coding was used to categorize the nodes. The number of colors was kept at a minimum to keep a high level of distinction (section 2.2.2). Yellow was used to signify domains, blue to signify services, and orange to signify interfaces. The node colors were chosen for visibility of labels and links. The background of the graph is white and the labels and links were black. Since the labels and links need to be easily distinguishable from both the background as well as the nodes, the nodes needed a somewhat light color too. Differentiating between different nodes should also be easy. With the colors chosen, the labels and links were visible on both the white background as well as all the node types.

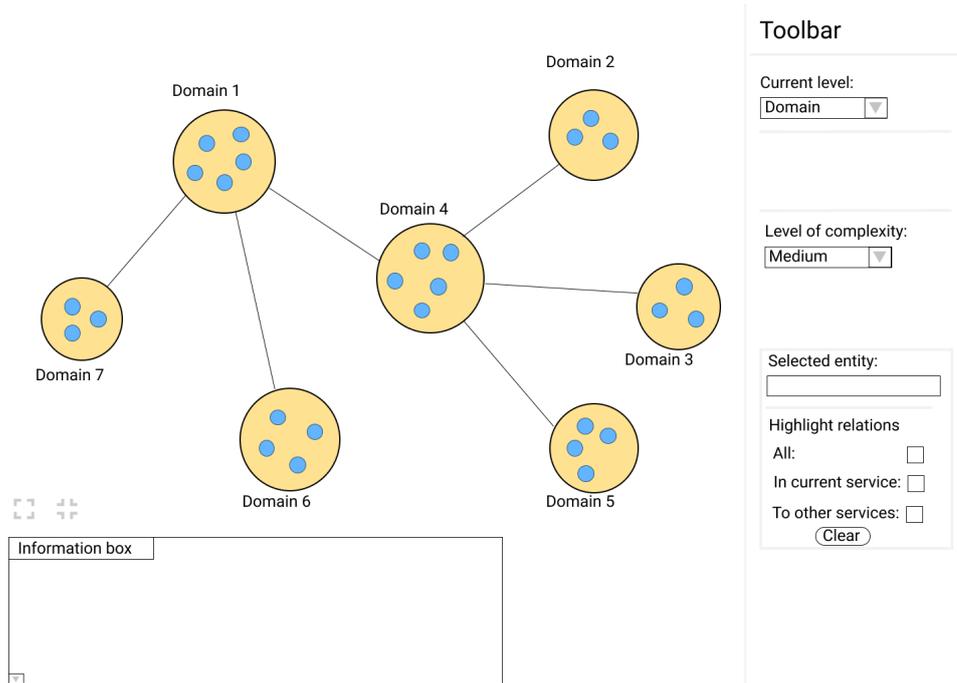
A dashed outline drawn around the nodes was used to show that the view had zoomed into a domain, see figure 3.8(b). Note that there is no dashed outline around the domain nodes in figure 3.8(a), as domain is the highest level. In the toolbar on the right-hand side, drop-down menus were used for changing the complexity level and the node level. By increasing the node level, the number of visible nodes increased. By increasing the complexity level, the number of links is increased. In figure 3.8(b) the node level is set to show services of the selected domain. The complexity level is set to high, meaning that links are drawn between the interfaces (orange) instead of between the services (blue).

Second iteration

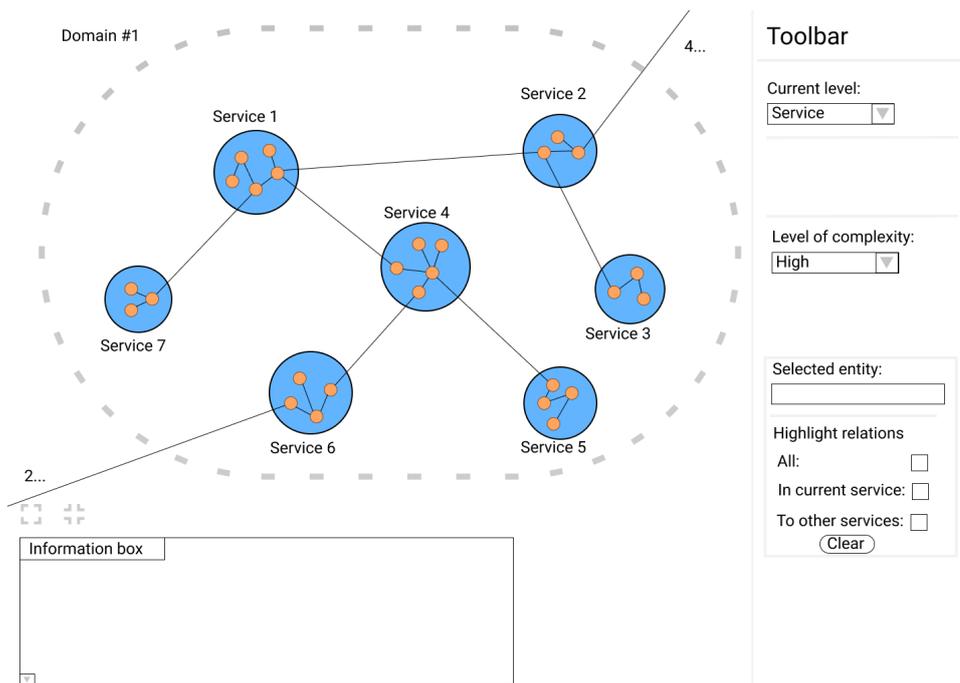
After a demonstration of the prototype from iteration 1 to one of the end-users where the prototype was shown via screen sharing, as well as an in-depth analysis of the interactive flow, a number of critical flaws were identified:

- It was unclear how the user would navigate between different services or interfaces.
- Using one drop-down menu for changing the number of visible nodes and another drop-down menu for changing the complexity (changing how many links were to be shown) was confusing.
- In a zoomed-in view, the user would have to zoom out to see links drawn outside the view. See figure 3.8(b). A possible solution to this was to allow the user to double click on a link going out of sight, update the view, and show the node that the link was connected to. However, this solution would not give the desired overview of the system and possibly confuse or even annoy the user (section 2.1.5).

A new design was developed. The main difference in this design, compared to earlier versions, was how nodes were hidden and the interaction to make more nodes visible. In the new



(a) Domain level with medium complexity.



(b) Service level with high complexity.

Figure 3.8: Mid-fi prototypes iteration 1. (a) In domain level, with medium complexity, displays the domain nodes (yellow), as well as the domains' services (blue). (b) A zoomed-in view of domain 1. The services (blue) and the services' interfaces (orange) are shown with complexity level set to high.

design, initially, only the domain nodes are shown. If the user wishes to see a particular domain's services, they can click on it, causing it to expand and the services to render. In figure 3.9(a), the user has clicked on "Domain 4". The node expands, and its services are rendered. In figure 3.9(b), the user has clicked on "Domain 2". Note that the links are automatically redrawn. Instead of being drawn between the domains, they are now drawn between the services. Analogously, if the user clicks on a service, the service expands, its interfaces are rendered, and the links are redrawn. To remove subnodes, the user clicks on the node containing the subnodes, and the reverse process to what is described above occurs.

This design turned out to be easier for the user to understand. It gave the user more control over where the graph's complexity should be increased and allowed the user to overview the whole graph while seeing more details of specific nodes.

Third iteration

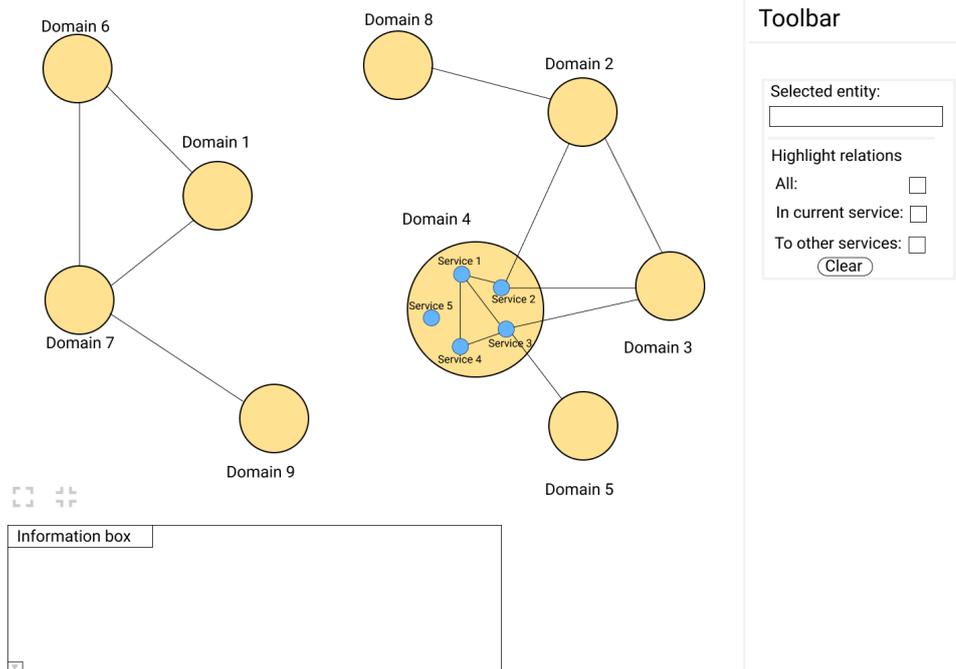
In the third iteration, the highlighting functionality was redefined and improved. In the previous version of the prototype, the user had to select a node and then highlight the selected node and its links using the toolbar options. With the changes of design in iteration 2, it became unclear how the user should select a node since clicking on nodes now made them expand. The improved highlighting function was defined to select and highlight a node and its relations by holding down Shift and clicking on the node, see figure 3.10. If a user selects a node with subnodes, all the subnodes and their relations will be highlighted.

At the request from our end-user, functionality for showing previous and upcoming versions of the data was added, as well as functionality for searching for specific nodes in the graph. See appendix C more pictures.

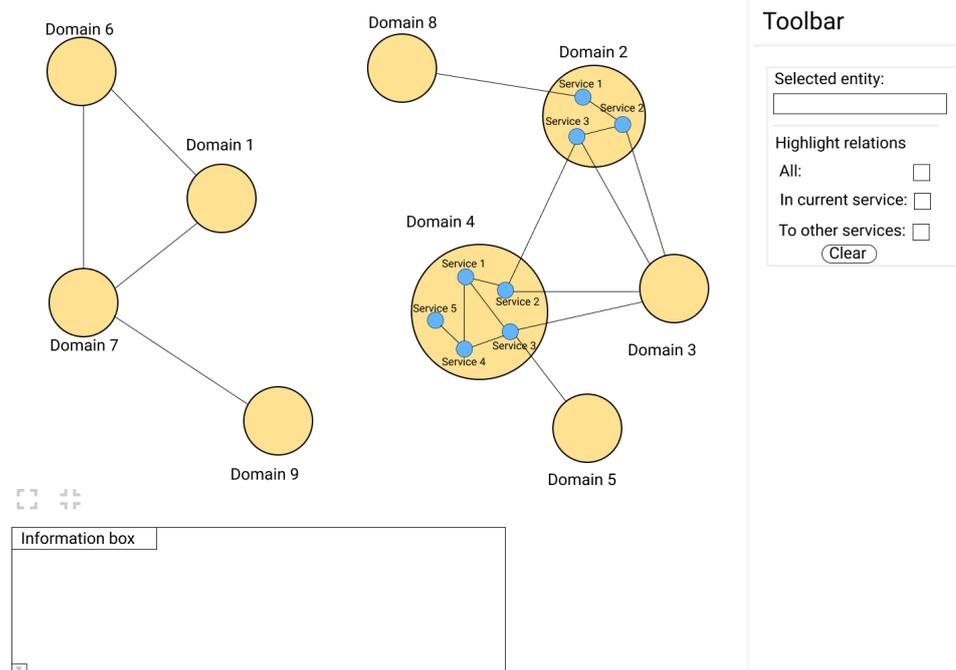
3.4 Implementation

The aim of the implementation phase was to, by using an agile approach (section 2.5.3), develop a tool based on the prototypes from section 3.3. As when the prototypes were developed, one end-user was involved during the implementation phase, allowing him to give feedback and suggest changes in both functionality and design. The implementation was done as a single-page web application written in HTML, JavaScript, and CSS. Compared to the lo-fi and mid-fi prototypes, the implemented version gave the user a more comprehensive understanding of how the interactions would work. We had sprints every week in which we tried to reach whatever implementation goals we had set for that week. Following an agile development approach, issues discovered during the sprint were discussed during our next meeting. Due to this format, the design changes did not arise in an as clear and concise manner as they did during the prototyping phase. Instead, changes were made when decided necessary and in consultation with our user.

One of the requirements was that the implemented tool was supposed to work on generic data, meaning that the graph's appearance would differ depending on the input data. For computing the nodes' positions, we utilized a library called D3 (section 3.2.2).



(a) The user has clicked on Domain 4.



(b) The user has clicked on Domain 2.

Figure 3.9: Mid-fi prototypes iteration 2. (a) The user has clicked on Domain 4, causing it to expand and its services to render. (b) The user now clicks on Domain 2, causing it to expand and its services to render. The links are redrawn, showing dependencies between the services instead of the domains.

The following text describes the methods used to implement the tool shown in Final design (4). We had no previous experience with the D3 library and limited experience in web development. Because of that, 2 weeks were first spent on familiarizing ourselves with D3 and basic web development techniques.

3.4.1 Setup

The first step was to implement all necessary functionality for handling the data representing the graph, which included:

- Parsing data and being able to convert it into objects representing nodes and relations (links) with necessary attributes.
- Relevant data structures and functions for finding specific nodes, their subnodes, supernodes, and related nodes.
- Manipulation of existing nodes and links for updating position, radius, relations, and subnodes, as well as being able to remove and add new nodes and links.

The next step was to implement a minimal HTML document structure with SVG containers for the graph elements. We used three SVG container elements, where the first one was used to contain the nodes, the second was used for containing the links, and the third was used to contain the labels. Since the drawing order of SVG elements is determined by the order in which they appear in the DOM, this structure made it easy to draw the nodes furthest back and the links and labels on top, simply by adding the elements to their respective container.

For mapping the data, internally representing nodes and links, to SVG elements that can be added to the DOM, we utilized D3's update features. Using them, D3 keeps track of how many new SVG elements should be added to the DOM and if any existing elements should be removed or updated.

For computing the nodes' position, we used D3's force simulation, described in (section 3.2.2). The main requirements we had on the graph layout were that related nodes should be positioned close to each other, and there should be as few edge crossings as possible. There should be sufficient space between the nodes for the labels while not spreading the nodes too far away from each other.

3.4.2 Initial position of domain nodes

Initially, when the web application is loaded, only the domain nodes are rendered. At this stage, our internal data representation of the nodes only contains the domain nodes, and the initial simulation should therefore be applied to all of them. With the requirements defined above, we found the best results using the following forces:

- **forceLink** with a strength of 1. This force pushed related nodes closer to each other, necessary for reducing edge crossings, as well as edge distance.
- **forceCenter** pulling all nodes toward the center of the screen without affecting the nodes' relative position.
- **forceX/forceY** used with both x- and y-coordinates set to 0. This force reduced the spreading of the nodes, pulling them closer to each other even if they were not related, making the graph more compact.
- **forceCollide** that, for every node, was set to the node's radius + 100. This ensured that there would be enough space between the nodes.

The effect of applying these forces to the nodes are shown in figure [3.11](#).

3.4.3 Initial position of subnodes

Finding the initial positions of the subnodes was not as straightforward as for the domain nodes. As seen in our prototype, the subnodes should be placed inside their supernode. Each domain's subnodes can be viewed as a separate network, optimized similarly to the domain nodes. For this, D3's force simulation was used, applied only on the subnodes before they become visible in the graph.

The first step was to increase the supernode's radius to make more room for the subnodes, as described in section [3.4.4](#) below.

A new set of forces was then applied to the subnodes. Note that these forces would not affect any other nodes in the graph that had already been rendered. The forces used were:

- **forceLink** that was set to 1 if the link was connected to another node outside the supernode. Otherwise, it was set to 0.5. This caused links connected to nodes outside the supernode to be pulled towards the supernode's edge, closest to the outside node, reducing both link crossings and link distance.
- **forceX/forceY** was used when there was only one subnode and set to the center point of the supernode, causing the single subnode to be centered.
- **forceCollide** that, for every subnode, was set to $(\text{the node's radius}) + (\text{the supernode's radius}) / (\text{number of subnodes}) + (\text{number of dependencies the subnodes have}) * 3$. This ensured that the nodes were placed with sufficient space between them while taking into account that the available space was determined by the size of the supernode.

As described in section [3.2.2](#), D3 uses a simulation that works by iteratively applying the forces on the nodes for a number of steps. The simulation has no default functionality for

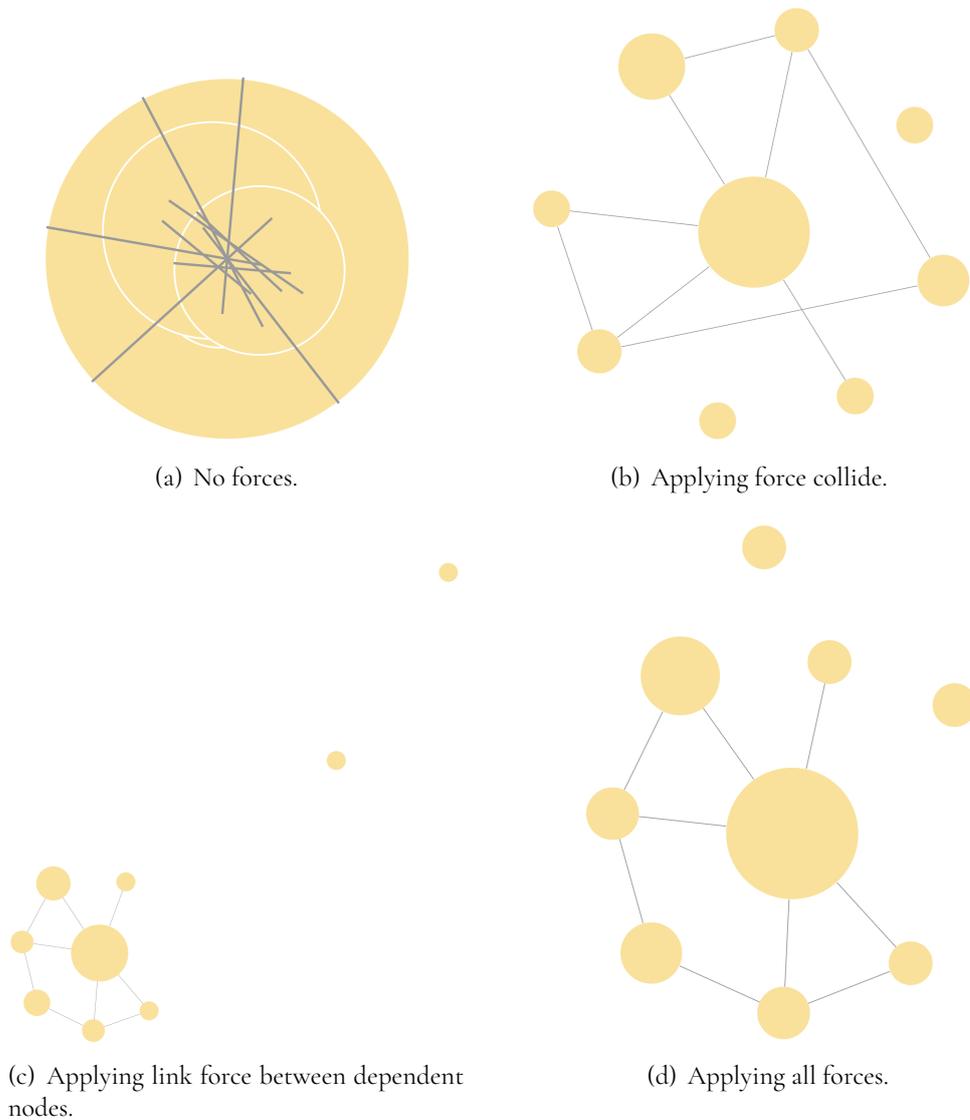


Figure 3.11: In (a) the domain nodes are drawn without any forces. The grey lines are links between dependent nodes. In (b) force collide is applied, separating the nodes from each other. With link force (c), dependent nodes remain close to each other, while other nodes are position farther away. In (d) all forces are applied.

defining an area in which the nodes should be placed. Thus, we needed a way to alter the simulation's behavior without affecting the simulation's purpose of finding an optimal placement too much. To bound the subnodes to remain inside their supernode, algorithm 1 was applied on every subnode in each simulation tick. This algorithm moved every subnode that, by the simulation, was positioned outside the supernode's edge to be moved to the farthest away position they could have within the supernode's area. Since algorithm 1 was applied in every simulation step, the simulation could take these sudden position changes into account when the simulation continued.

Algorithm 1 Constrain subnode position

```

1: for each subnode do
2:    $d \leftarrow$  distance between subnode's and supernode's centers.
3:    $limit \leftarrow$  supernode's radius * (1 - margin in %) - subnode's radius.
4:   if  $d > limit$  then
5:     Update subnode's position, placing it at  $limit$  from the supernode's center with-
       out changing the subnode's angle relative to the supernode.
6:   end if
7: end for

```

The effect of applying algorithm 1 and the forces mentioned above to are demonstrated in figure [3.12](#)

3.4.4 Expanding and moving of nodes

Whenever the user clicks on a node in the graph, its subnodes should become visible. Before any subnodes are displayed, the clicked node must expand to make room for the subnodes. How much the node should expand depends on its subnodes and must be sufficient for the graph drawing algorithms to find optimal positions for the subnodes according to previously defined requirements. At the same time, the node should not expand more than necessary, leading to the node taking up more screen space than it has to. The expansion of the clicked node's radius depends on the sum of all its subnodes' radiuses and the number of dependencies (number of links) the subnodes have.

In many cases, when a node expands, it will become too close to other, surrounding nodes at the same depth in the graph, so-called sibling nodes (figure [3.13](#)). Using algorithm 2, we detect if the expanded node has become too close to any of its sibling nodes. If it has, all the sibling nodes are moved away from the expanded node the same amount as the node expanded while preserving the relative angles between the nodes.

If the expanding node is a subnode, we use the same method described above to move the sibling nodes if the node has become too close. However, since subnodes' positions are restricted to their supernode's area, we first must increase the supernode's radius and also move its siblings if they becomes too close. An example of this is if the user clicks on node 1 in figure [3.14](#) the following steps would take place:

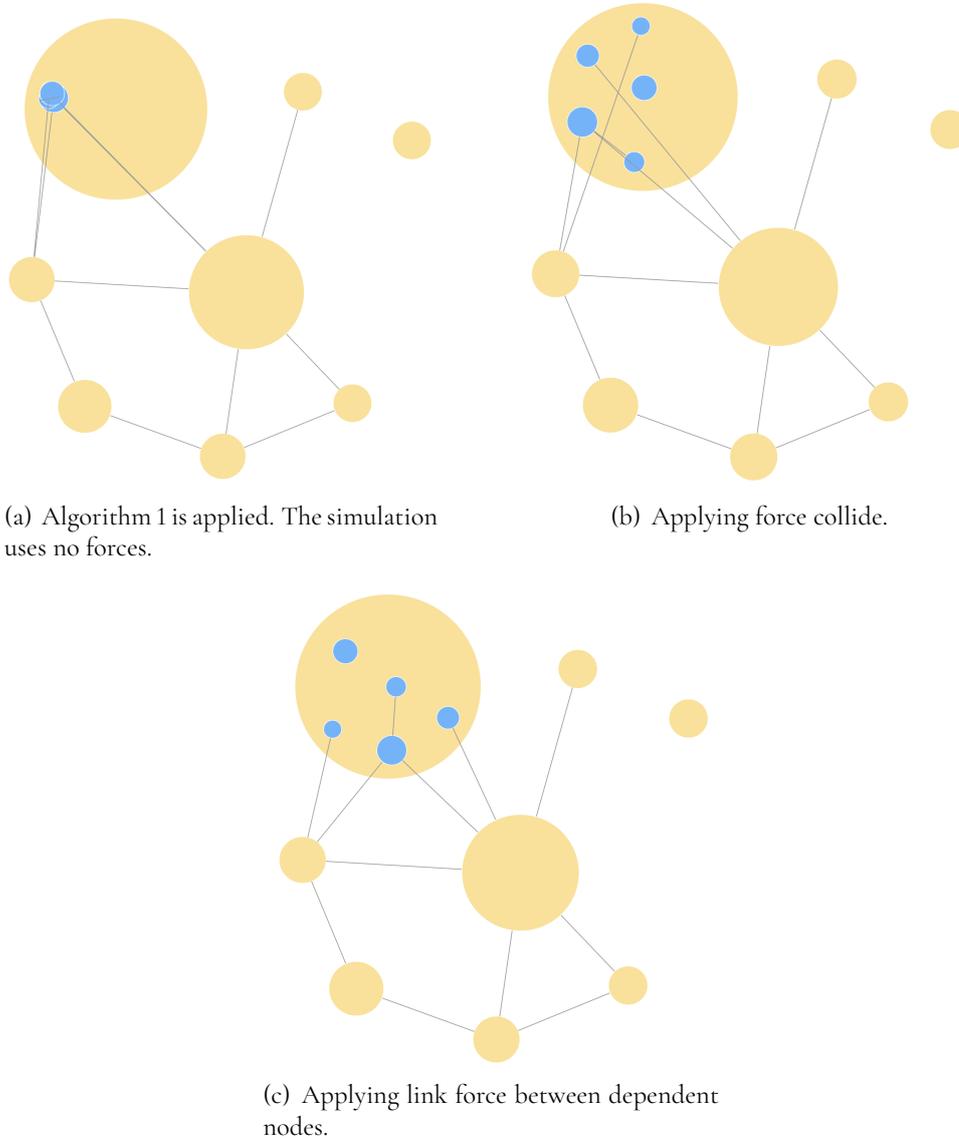


Figure 3.12: In (a), the subnodes (blue) are drawn without any forces. Algorithm 1 is used to ensure that the nodes are positioned within their supernode. In (b), force collide is applied, separating the nodes from each other. With link force (c), dependent nodes remain close to each other. Nodes with links connected to nodes outside the supernode are pushed toward the supernode's edge, minimizing the link distance and reducing link crossings.

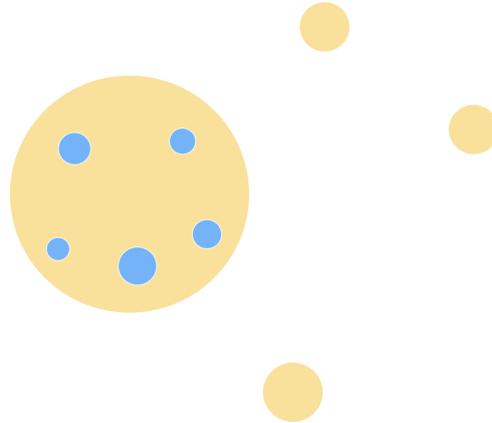


Figure 3.13: All the yellow nodes are at depth 0 of the graph since and are therefore siblings to each other. The blue subnodes are at depth 1. All subnodes within the same supernode are considered to be siblings to each other.

Algorithm 2 Detect if expanded node is too close any of its siblings

```

1: for each sibling node do
2:    $d \leftarrow$  distance between sibling node's and expanded node's centers.
3:    $sum\_radiuses \leftarrow$  sum of sibling node's and expanded node's radiuses.
4:   if  $d \leq sum\_radiuses + margin$  then
5:     Collision detected.
6:     return true
7:   end if
8: end for

9: No collision detected.
10: return false

```

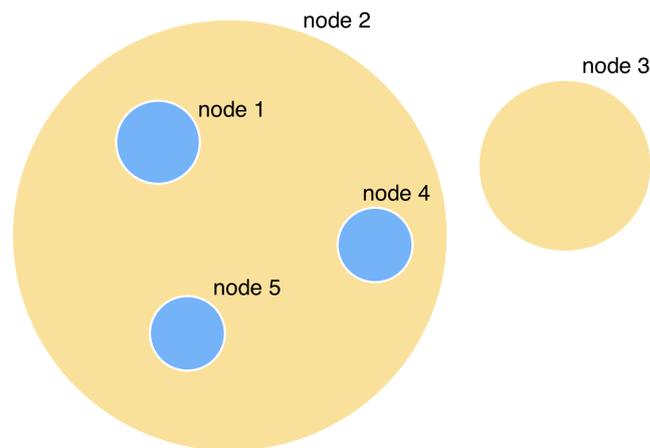


Figure 3.14: If the user clicks on node 1, it expands, and its subnodes are rendered. When node 1 expands, node 2 might need to expand and make more room for its subnodes. When node 2 expands, it will become too close to its sibling node (node 3), which will have to be moved as well.

1. Node 1 expands in order to make room for its subnodes.
2. If node 1 becomes too close its sibling nodes (node 4 and 5), the siblings node must be moved, while still remaining inside node 2. To make this possible, node 2 has to expand as well.
3. When node 2 expands, its sibling (node 3) must be moved so that node 2 and 3 does not collide.
4. Node 4 and 5 are moved the distance corresponding to the expansion of node 1.

Depending on the nature of the data, there could be any number of supernodes to a given subnode, and thus must this procedure be repeated for all supernodes until depth 0 has been reached. An example of this is shown in figure [3.14](#). With this in mind, it is easy to realize that any node's position can be changed several times, and any node with subnodes can be expanded several times. If the user clicks on a node that already has its subnodes displayed, the subnodes are removed. When this happens, both the position of surrounding nodes and node expansions should be restored to exactly how they were before the clicked node expanded. To achieve this, we kept track of how every expansion of nodes affected other nodes' radiuses and positions and linked that data to the clicked node's id. It was then easy to reverse both positions and radiuses when needed.

3.4.5 Labels

Our early implementations placed the labels on the right-hand side of every node. This naive solution worked well, and the number of labels overlapping was kept at a minimum as long

as there was sufficient space between the nodes. However, having much space between the nodes to prevent overlapping labels did not scale well since some nodes had to be very large, making it hard to overview the graph. Instead, a modified version of the D3-labeler library (section 3.2.3) was used. The library implements a simulated annealing algorithm, optimizing the label placements with respect to surrounding labels, as well as surrounding nodes. See figure 3.15

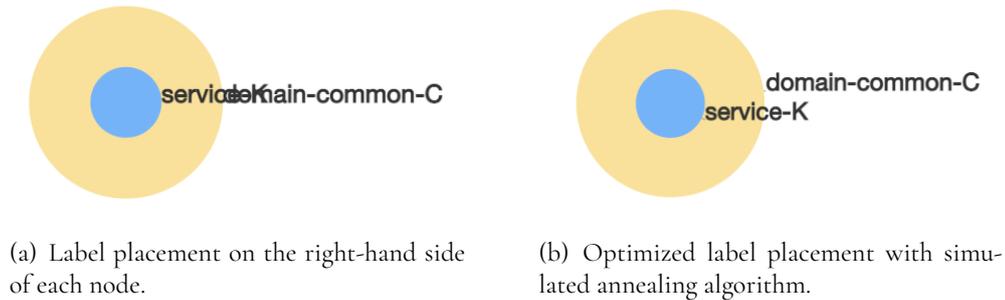


Figure 3.15: In (a), labels are placed on every node’s right-hand side. The result of this naive approach is dependent on the nodes’ sizes. In (b), a simulated annealing algorithm is used to optimize the labels’ positions, mitigating the risk of labels overlapping.

The algorithm is not perfect and does not consider labels overlapping links, meaning that the labels will still be hard to read in some cases. To remedy this, we implemented functionality allowing the user to move the labels manually. A thin line between each label and its respective node was used to clarify belonging, see figure 3.16.

The font size was set to be relative to the zoom scale to increase labels’ readability. When the user zooms out, the font size is increased and vice versa when zooming in.

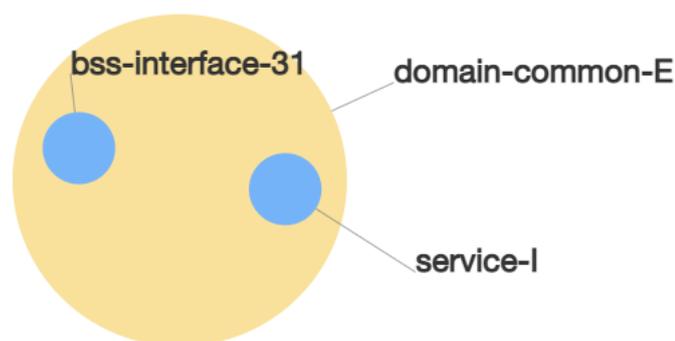


Figure 3.16: Labels that have been manually moved. A thin line shows each label’s belonging.

3.4.6 Other implementation details

Zooming functionality was implemented, allowing the user to get a closer view of smaller sections of the graph.

The highlighting was implemented according to the prototype, allowing the user to select a single node, reducing all nodes' and links' opacity not directly related to the selected one.

Since the implementation is based on a simulation of physical forces, the result will vary and, in some cases, produce graphs where some of the nodes' positions are not fully optimized. This lead to the need to allow the user to move nodes' positions manually. By holding down shift, the user can click-drag on any node to move it. This functionality is meant to be used only when necessary to increase readability and reduce clutter.

The toolbar and infobox were added to the web page in accordance with the design of our mid-fi prototype. After the visual elements were added, functionality to make the visual elements interactive were added.

Chapter 4

Final design

In this chapter, the implemented tool is presented. The results are divided into two parts, one describing the graph and one describing the toolbar. After several small changes, meetings with feedback and internal discussions on how to solve issues with the tool we had arrived at our final design. Since we designed as we developed, the final design is also the final version of the tool. Discussion regarding design choices and such is found in chapter 5.

The figures in this chapter, showing the final results, use the same data as was used in figure 1.1. However, the tool is implemented such that it is possible to visualize other data in the same format.

Additional pictures of the final result can be seen in Appendix C, section 10.3.

4.1 The graph

When the graph is initially loaded, only domain nodes are displayed, see figure 4.1(a). Dependencies are shown as straight lines with arrows pointing in the direction of the dependency. The user can decide which nodes they want to explore by clicking on them, causing the clicked node's subnodes to be added to the visualization, see figure 4.1(b), thus only increasing the graph's complexity on the parts that the user chooses. If the user clicks on an already expanded node, like the yellow sections of the expanded nodes in figure 4.1(b), domain-product-B and domain-product-E, the domain nodes are collapsed and the graph goes back to the state of figure 4.1(a). This implementation makes it easy for the user to revert to any previous state

if they want to, promoting the user to explore the graph.

In figure [4.2](#) the highlighting functionality is illustrated. In figure [4.2\(a\)](#), the user has selected the service node (blue). Its contour becomes black, and the opacity is lowered for all nodes and links not directly related to it. In figure [4.2\(b\)](#), the user has continued by clicking on some other nodes without changing which node is selected. The nodes and links added to the visualization in figure [4.2\(b\)](#) will only be included in the highlighting if they are directly related to the selected node.

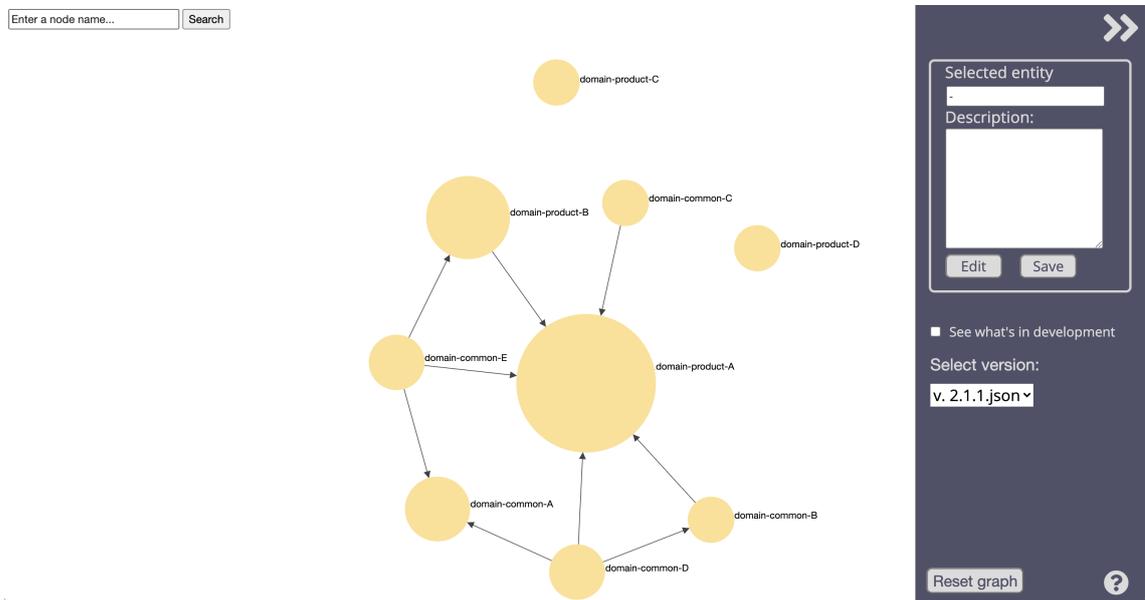
Figure [4.3](#) illustrates the search functionality. When the user searches and selects a node from the suggestion list, see figure [4.3\(a\)](#), the node will get added to the graph, if not already present, as well as selected and highlighted, see figure [4.3\(b\)](#).

4.2 The toolbar

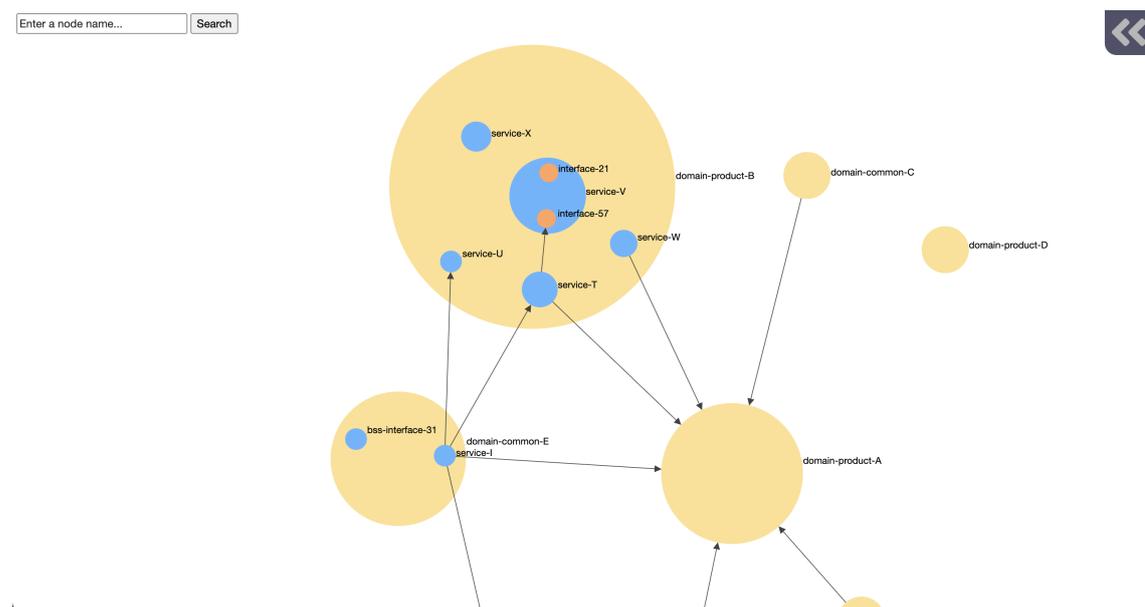
The final toolbar is as seen in figure [4.4](#). In the top right corner there is a button for hiding the toolbar, should the user want to see only the graph. Under this, a section related to which entity is selected is found. Here we have a text field that will contain name of the node that was clicked last, in figure [4.4](#) service-I is selected. There is also a description field where the user can edit and save changes it makes to the selected nodes description attribute, an attribute the company might add to their data. There is a checkbox the user can cross and uncross to see and hide what is in development, meaning changes in the data that are in the making such as nodes being added or disappearing. Under this there is a dropdown menu that the user can select file versions to load. Selecting a file from the dropdown will redraw the graph based on the data in the newly selected file. Currently the file v.2.1.1.json is selected. In the bottom left, reset graph button is placed. This button will reset the graph to its starting state, with all nodes collapsed so that only domains are seen. At the bottom right corner there is a button with a question mark which is a help button. Clicking this triggers an overlay where instructions on how to use the system is listed, see figure [4.5](#). In the top left there is a search field and a search button, which the user can use to search for specific nodes. As the user types, suggestions based on the input will appear below, see figure [4.3](#).

The functionality of following elements is not implemented:

- Select version.
- See what's in development.
- Editing of node descriptions.

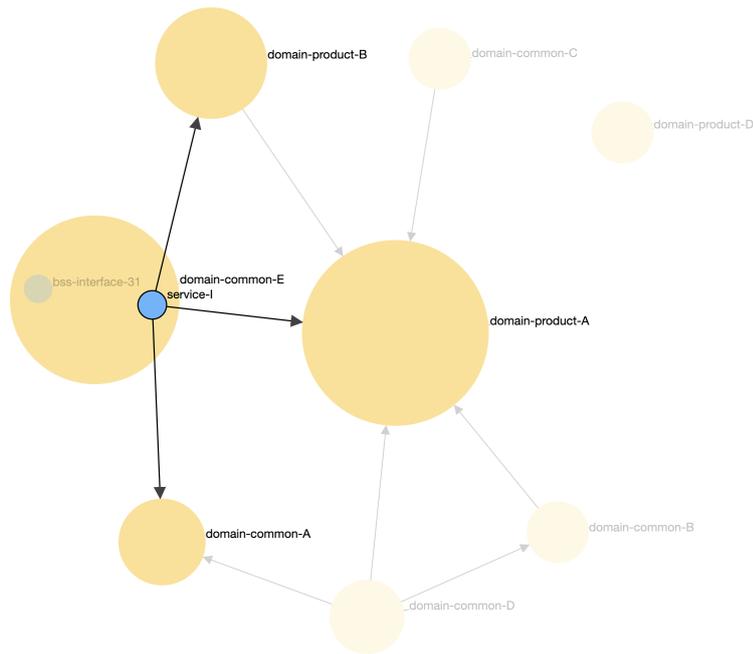


(a) Initial appearance.

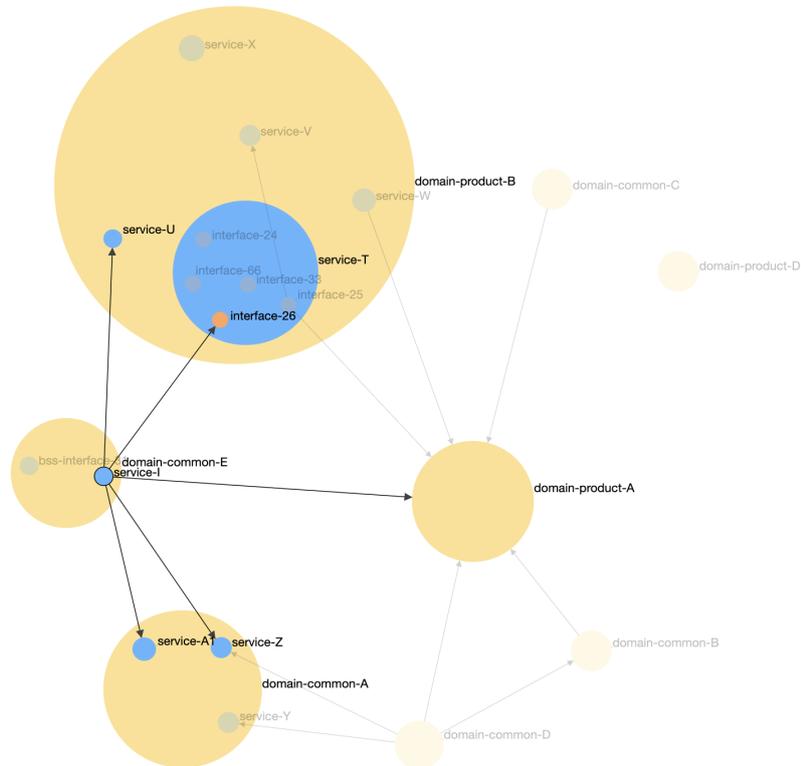


(b) Example of user interaction.

Figure 4.1: (a) The initial appearance. In (b), the user has clicked on two domain nodes (yellow), as well as a service (blue). The user has also hidden the bar on the right-hand side, as well as zoomed in on the nodes.

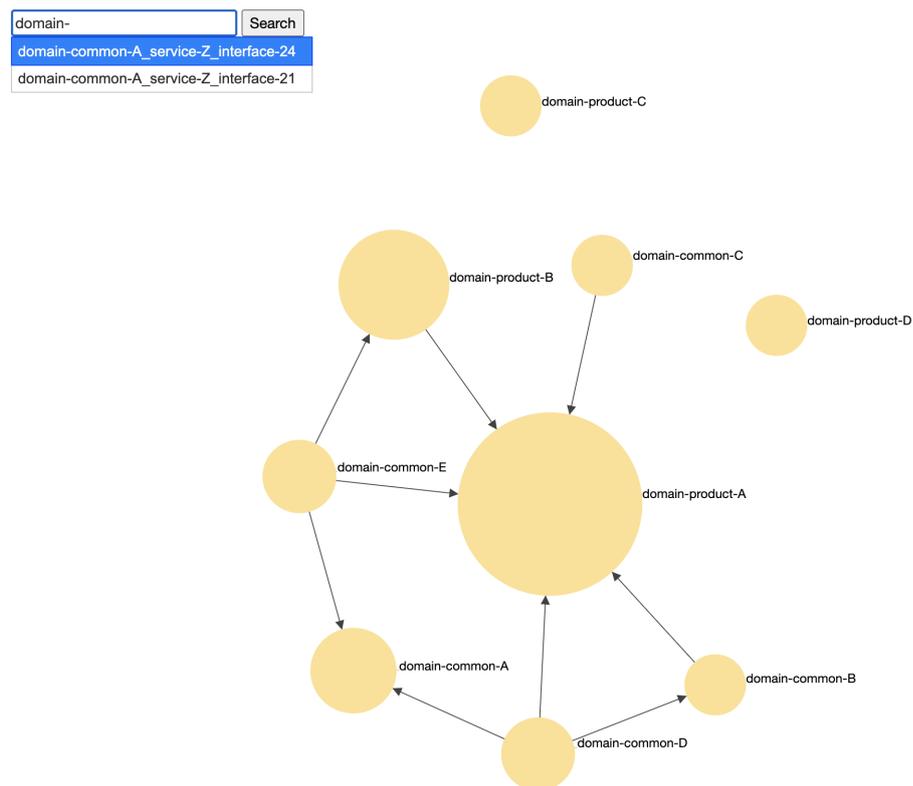


(a) User selecting a service. The selected service-I is shown with a bold contour. Nodes not directly related to the selected node have their opacity lowered.

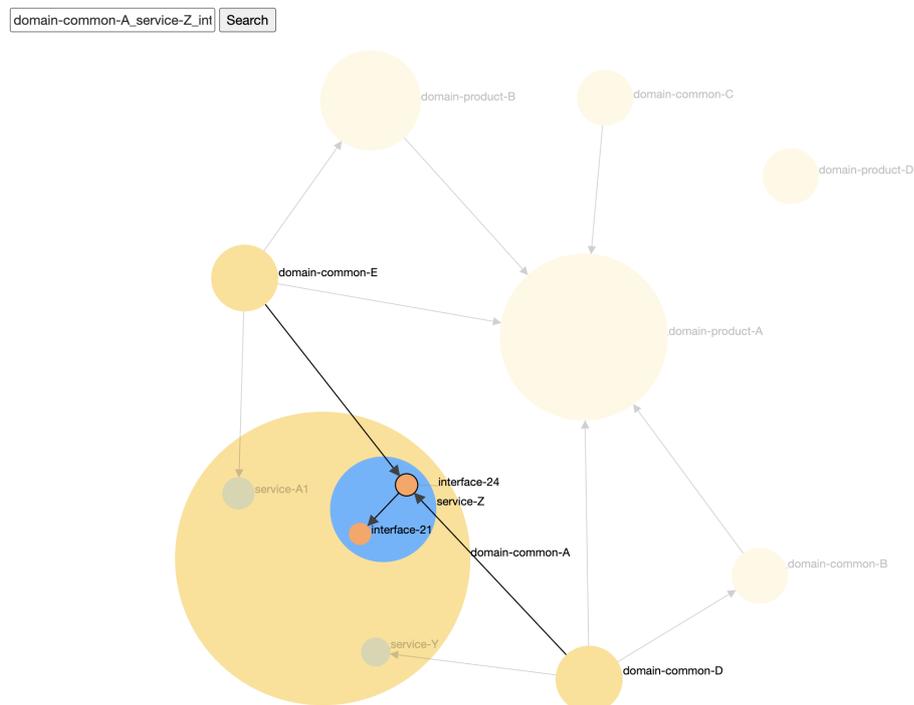


(b) The user clicked domain-product-B and its subnode service-T, and domain-common-A. The selection on the service from (a) has not changed. Now subnodes of the clicked domains, that are related to service-I, are also highlighted.

Figure 4.2: Illustration of the highlighting.



(a) The domain nodes are shown. Suggestions appear under the search field as the user types a query.



(b) A search has been done. The node that was searched for is highlighted and its supernodes have been expanded.

Figure 4.3: The tool's search functionality is illustrated.



Figure 4.4: An illustration of the final toolbar.

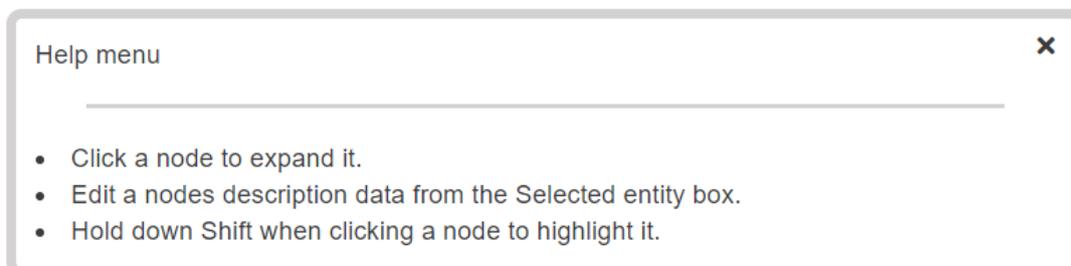


Figure 4.5: The help overlay triggered by clicking the help button in the bottom right corner.

4.3 Final test

The final user test was done remotely. The users involved were the two supervisors from Sinch, one of whom is a System Architect at the company. Microsoft Teams was used to demonstrate the final version of the tool. The test was done by showing and telling the users the different ways to interact with the tool. This was done with Microsoft Teams' screen sharing functionality. Scenarios were used to give the users insight in when the functionality could be used. For example, for showing the search functionality the scenario of "not wanting to look through different node layers to find an interface you know the name of, but not its domain" was described. After the scenario was described, the search function was used. The other parts of the tool was shown in a similar way. They were happy with the result and thought it looked good. They gave us some feedback on improvements we could make, and also imagined a few use cases where the tool could be utilized. While some functionality remained unimplemented they still believed it would be a good starting point for future development of the tool within the company.

Chapter 5

Discussion

In this chapter, we discuss the obtained results and compare them to the requirements and requirements defined in section [3.1.3](#). Finally, we propose how the project can be future developed.

5.1 Method and process

We used a loose version of agile software development, in particular the framework Scrum, where we had iterative design processes. Other than our prototype our requirements were quite open-ended, except for a few vital basic functionalities. Our supervisors at Sinch gave us a lot of freedom in how the final product would look. Smaller changes in the design were done during development without including the end-users, but they were involved in all major design changes. Meetings were held where they could give feedback for those. This model fit our work from home style of work very well, and let us work independently without a problem. For the prototyping we had two major iterations, one for the first version of the design and a second one for what became the final one. During the implementation we had continuous communication and coordinated with each other once per week, where we discussed the design choices internally. This is more described in the **Development** section. While the development itself was quite flexible, we had an outline for the whole project as is seen in appendix A.

5.2 Development

We had many discussions regarding the exact details of how the graph would behave. As the project progressed, we realised that some aspects of the graph needed more thought put into them. The answer of which interaction is the best one for a specific problem is often far from obvious, and we sometimes paused the development to focus on making new prototypes on specific interactive elements to make sure we picked the right one.

With our development style we could quickly adapt to new situations. This was very useful when unexpected bugs or design changes arose, allowing us to re-prioritize or divide the workload such that the development did not come to a standstill. It was also useful for the cases where we had added or removed features. If we had a very detailed plan, we would have had to rewrite our plan to adapt to those changes, but with the agile approach we could instead do as we saw fit at any given moment.

The development took longer than expected and we had to split our workload so that one of us focused on report writing, while the other finished up the code. This could possibly have been avoided by more rigid planning.

The toolbar and overlay was harder to implement than anticipated. We did not have much previous experience writing webpages other than smaller personal projects. When programming in environments you have a greater knowledge in, a big part of problem solving is searching the webs for similar problems. Limited knowledge of the technical terms in HTML and CSS made this harder. It also took some time to understand and utilize development tools such as the debugging console in browsers. We underestimated these aspects of web development in our planning however which lead to lost time. Another aspect of the web development that could have been done differently is to use JavaScript frameworks such as React or Angular, or CSS frameworks such as Bootstrap. There would have been some advantages to use those, such as making reusable UI components in React or using design templates in Bootstrap. Bootstrap was used for a very small number of elements and could have been utilized way more.

5.3 User centered design

One of our goals was to involve users in the development. This section is a summary and reflection of how it went.

- The users, our supervisors from Sinch, were highly involved in the start of the project where many discussions were held that helped us understand what kind of data we were to visualize, as well as what the end-users desired.
- After the startup phase meetings were held every 2-3 weeks. Our designs were shown and feedback was received. In preparation for those meetings, interview questions

about aspects of our designs could have been prepared, to get a deeper understanding of how the design was received.

- There could have been more communication from our side of things. The thesis was done almost completely from home. We had access to communication with one of our supervisors from Sinch via a personal messaging service, this could have been utilized more for short questions instead of us waiting to ask all our questions at the next meeting.
- Covid-19 affected us in that we worked remotely, which made proper user testing harder to do. Letting end-users test the tool remotely became hard when working with unfinished versions of our development. We could have been more proactive and made sure to upload stable versions that the end-users could interact with.
- That we chose to show and tell, instead of letting the users set up stable versions of the tool to interact with, reduced our insight in how well the tool actually would work when used, as well as how well it followed design principles. For example, determining how well discoverability was implemented is hard when the user is not able to naturally interact with the tool.
- For receiving feedback on the design, online forms could have been sent to a larger sample group than the two users we had communication with from Sinch.

5.4 Final design

In this section we will first discuss the result and design choices that were made for the graph and the toolbar respectively. After this we will discuss how well we have or have not reached the requirements we set up for ourselves.

5.4.1 The graph

Using D3's force-directed algorithms, simulating physical forces between the nodes, we could produce a graph with good node placement (section [2.3.3](#)) without any previous special knowledge in graph theory, which fitted this project well since the focus was on visualization techniques and interaction design. However, a downside with force-directed algorithms is that small changes in the input data can result in large differences in the graph's layout. Another problem similar to the one described above can be seen in figure [10.8](#) in appendix C. Depending on the graph's state, there will either be one or two links between "domain-product-B" and "domain-product-A". Since the implementation uses forces between the links to pull connected nodes closer to each other, this affects the subnodes of "domain-product-A". These two problems are probably the most significant disadvantages of force-directed algorithms and results in considerable layout inconsistency. This inconsistency prevents the user from memorizing nodes' positions (section [2.1.6](#)) and will most likely lead to frustration.

The most significant difference between the implemented graph and the prototypes is that the implemented graph is not perfect. In some cases, depending on the data, there will be link crossings, and nodes positioned inconveniently. During the prototyping, we never considered this, primarily since we had no previous experience of graph drawing and did not know how well the actual implementation would perform. Allowing the user to move nodes and labels manually was a simple yet useful solution to the problem. However, manually moving the nodes will affect the outcome from the force-directed algorithms, which could lead to unexpected results.

The biggest challenge during the implementation was to construct a generic solution for computing the subnodes' positions. What made this problem difficult was that they had to be placed inside their supernode. In other words, there was not unlimited space for the algorithms to utilize when computing the nodes' positions. For solving this problem, we tried a number of approaches in addition to the one we used in the implementation. The most promising method was to balance forces pulling subnodes to the center of the supernode, and repulsing forces between the subnodes. Finding a balance between the forces that worked well for different supernodes with varying subnodes turned out to be complicated. Another problem with this approach was that when there were many subnodes, many of them were positioned close to the center, resulting in many link crossings.

As described in section [3.4.4](#), the surrounding nodes' position would have to move when the user clicks on a node, making it expand. When moving nodes that already were displayed on the screen, it had to be done predictably, making it easy for the user to keep track of the nodes' new positions. The implemented method moved all nodes an equal distance, meaning that the relative position between them remained the same. Another approach we experimented with used a collision force on the expanding node, causing only the nearby nodes to be moved. The advantage of this method was that fewer nodes were moved whenever a node expanded. However, depending on the user's interactions, the graph's overall appearance could be significantly altered, and therefore, this approach was discarded.

5.4.2 The toolbar

Some design choices behind the visual elements are discussed here. Some of the functionalities that were planned for the toolbar were not implemented due to time constraints. The visual elements are there however and as such parts the toolbar can be seen as a mock up.

During the process of implementing the toolbar, a few things changed from the original design. When actually developing and seeing the webpage in action, you get a deeper understanding for what would be useful, what is unnecessary and general improvements. Similar to the thought process when designing the graph, we wanted the toolbar to look good as to help make the user feel good (section [2.1.3](#)). At the end of the development the toolbar was changed visually, and different colors were used and the different elements were refined. Feedback (section [2.1.2](#)) was added to all buttons when the user hovered over or clicked them.

Since a lot of functionality already exist in web browsers and the D3 library, some of the

elements in our mid-fi prototype were removed, such as the zoom button and enter/leave fullscreen buttons. As they were redundant they would only increase display clutter (section 2.2.5).

While designing in such a way that the user does not need instructions, but rather in a way that invites the user to learn the system by exploring (section 2.1.4) and or intuitively understand it immediately is the ultimate goal, that is not always feasible. Some of the interaction would be hard to find by just exploring and discovering, such as "hold down shift while clicking a node to highlight it". As such, a help button which triggers a pop-up box with instruction was added.

Aesthetically speaking, a lot changed from the mid-fi prototype to the final design. The same core elements were kept but a lot changed in the details of their implementation. Rather than having lines separate the toolbar from the graph, we let the toolbar have a background color different from the background color of the graph. A dark color was chosen for the toolbar, making the graph and it be clearly separate elements due to contrast differences.

For the search functionality, a different solution was used than originally planned. Instead of having a search field directly in the toolbar, we decided to have it visible at all times instead, see the upper left corner of figure 4.1(a). This will make it more accessible to the user, should the toolbar be hidden. Another tested solution was to have search button in the toolbar, which when clicked triggered an search menu that floated over the graph. We instead chose to place it directly in the graph. As searching this could be a frequently used feature, it should be easy to access, thus we removed the extra step of selecting it via the toolbar.

The infobox was removed. Now the information that was to be displayed there instead is displayed in the toolbar, under the "Selected entity". Now, instead of it being just the description of a node being displayed, there are now buttons for enabling the user to edit and save the edit, updating the data file. To move the description and make it editable was suggested from one of our end-users during a demonstration we held for the company. The functionality for this was not implemented due to time constraints but the visual elements for it are there. Due to time constraints implementing a way of seeing what is in development was not done.

5.4.3 The requirements

We will now discuss how our minimum requirements were fulfilled or not.

1. The user should be able to get a good overview of the data by using the tool. We chose the vague term "good overview" on purpose since we were not sure on exactly how we would visualize the data. This vagueness has the downside that it is hard to measure empirically, but one could argue that our method of hiding unwanted information and giving the user control over which nodes are expanded does give a more comprehensible overview. Too many nodes or visual elements on screen makes a graph harder to understand. Our graph has lessened the number of visual elements. We have also have zooming in our graph, as well as incremental exploration and navigation, both which are suggested methods to mitigate the difficulty of

comprehending very large systems (section [2.3.2](#)).

2. The user should have a high degree of control of how much information is shown at any given time. The user does have some control of what is shown. Again, we have no framework to measure "high degree of control" empirically. We can however present arguments for how we worked towards it. Similar to requirement 1, the zooming and incremental navigation gives the user some control of what is shown. We also have searching and highlighting. While the highlighting does not quite hide information completely, it may make it easier to discern which nodes are of interest, by the use of pre-attentive properties .

3. The user should be able to switch between different data files. It is possible to use other data to redraw the graph. The result may not be optimal and further improvement to the graphs forces may be needed. A solution where the program retrieves the new file from a server was not implemented due to time constraints.

Now we will discuss the other requirements, that the product should look aesthetically pleasing and that some graph interactions should be implemented. Regarding the looks, we have received verbal feedback from supervisors from the company that our solution looked good. We could have done more to verify this, such as sending out user evaluation forms where people gave their opinion on whether it looked good or not. The graph interactions we mentioned in our requirements have all been implemented - zooming is possible, filtering is possible by searching and highlighting is possible as well.

Our final product had managed to do many of the things we set out to do. Much of the information is hidden and the user has got the ability to interact with the tool in such a way that it can manipulate what is shown. Without proper user testing it is hard to measure how well we managed to implement and design for discoverability. The end-users can be expected to know how the data is structured, and may immediately think that nodes with specific names should contain subnodes related to the name. Despite this, it is not obvious that clicking a node will make its subnodes visible for a first time user. For the non obvious interaction there is a help button which the user can click, where it can see instructions for the system, such as the highlighting. There is a risk that the user never clicks this button and reads, nor finds out by testing new things, that holding down the shift-button and clicking highlights the node. For those cases, there will follow a readable manual that the users can read to get an introduction to the system, should the discoverability not suffice. An alternative solution to this would have been user onboarding (section [2.1.5](#)) that would show them the different ways to interact with the system. The number of end-users is not very large and as such the importance of discoverability is lessened, compared to if it were an app to be available to a very large number of people.

It is clear that for the minimum requirements we could have benefited from thinking through our formulations. While vague terms let us develop in a very flexible manner, it made some of the requirements quite vague. But on the other hand, there is a risk that more concrete terms would have limited our possible routes of actions.

5.5 Future development

Although the implementation produced a significantly better graph than the initial prototype from Sinch, figure [1.1](#), the result is not entirely satisfying. Some improvements can be made, where the most critical is to solve the problem where subnodes' positions are affected by the graph's state, as described in section [5.4.1](#). This problem exists because the algorithms use all the visible links in the graph for computing the subnodes' positions. In other words, every visible link in the graph will have an equal pulling force on the subnodes that they are connected to. A possible solution to this problem would be to use the same set of links in the algorithms no matter how many links there are visible in the graph. The set of links could be the initial ones seen when the application is first loaded. This would ensure that the links' pulling force on the subnodes remains constant.

Due to time constraints, functionality to display nodes representing subsystems in development and nodes representing previous versions was not implemented, although included in the prototype. This is also the case for functionality allowing the user to change the data input file. These would be valuable features for understanding how the system has evolved and why changes have been made. Another valuable feature would be to allow the user to add new nodes and links directly in the application and edit existing ones. Currently, the user has to make these changes in the input data file.

Chapter 6

Conclusion

The following chapter concludes this thesis by answering the research questions, and discussing the goals defined in section [1.3](#) and comparing them to the results.

6.1 The research questions

- **How to visualize a complex system architecture, without losing too much information?**

A way to do this is to hide data and let it appear by user interaction, letting the user gain a comprehension of the graph in an exploratory way (section [2.3.2](#)).

- **Which factors will affect a users comprehension of a node networks?**

The number of nodes, how edges are drawn and how labels are placed are three factors. Too many nodes and edges in small spaces can reduce comprehension (section [2.3.2](#)), edge crossings should be minimized (section [2.3.3](#)), labels should not overlap (section [2.4.2](#)).

- **How can an application that visualizes the data structure be implemented?**

One way is to parse the data and let every object represent a node with relations to other objects. Then, force-directed algorithms can be used for optimizing node placement and minimizing edge crossings. For placement of node labels a simulated annealing algorithm can be used.

6.2 The goals

Based on design and interaction principles, construct a prototype for effectively visualizing multi-layered data with relations. It should be clear which nodes are in the same group and which nodes are related to each other.

The prototype demonstrates how complexity can be reduced by hiding nodes that the user is not interested in and allowing the user to interact with the graph to gradually increase complexity, that is, visible information, only in areas of the graph that the user chose. Combining this with highlighting and zooming, the user can focus on specific nodes and their relations, reducing complexity even more. Using these functionalities, the user can explore the graph to gain information about the underlying data and its relations. Search functionality makes it easy for users with knowledge about the underlying data to quickly find a particular node. The prototype included a suggestion for incorporating previous and future versions of the graph with the current version, coding the nodes and links related to previous data with low opacity, and those related to future data with dashed lines.

As with many design-related matters, the nature of this goal is subjective, and there is no definitive way to measure if the suggested prototype has satisfied the goal or not. However, since we have included our supervisors from Sinch during the design process and got their approval, we believe this goal has been satisfied.

Implement a tool based on the prototype. The implementation should work for generic data, meaning that evaluating and selecting suitable graph drawing algorithms must be done.

An implementation of the prototype was conducted utilizing force-directed graph algorithms from the library D3. Although there is room for improvements, especially for subnodes' placement, all key functionalities from the prototype were implemented as a single-page web application. The implementation was done using HTML, CSS, and JavaScript. Nodes, links, and other graph elements were represented by standard SVG elements, such as circles and lines. Label placement was done using a simulated annealing algorithm, reducing label overlaps. A toolbar and other functionalities from the prototype, such as highlighting, searching, and zooming, were implemented.

Chapter 7

References

Arnowitz, J., Arent, M., & Berger, N. (2006). *Effective Prototyping for Software Makers*.

Azzam, T., & Evergreen, S. (2013). *Data Visualization, Part 1: New Directions for Evaluation*, Number 139 (1st ed.). Jossey-Bass.

Cambridge Dictionary. (2021, March 24). cognition definition: 1. the use of conscious mental processes: 2. the use of conscious mental processes: . Learn more. Cambridge. <https://dictionary.cambridge.org/dictionary/english/cognition>

Cohn, M. (n.d.-a). Scrum Task Board. Mountain Goat Software. Retrieved March 26, 2021, from <https://www.mountaingoatsoftware.com/agile/scrum/scrum-tools/task-boards>

Cohn, M. (n.d.-b). The Daily Scrum Meeting. Mountain Goat Software. Retrieved March 26, 2021, from <https://www.mountaingoatsoftware.com/agile/scrum/meetings/daily-scrum>

Computational Complexity Theory (Stanford Encyclopedia of Philosophy). (2016, July 20). Stanford Encyclopedia of Philosophy. <https://plato.stanford.edu/entries/computational-complexity/>

Engelbrechtsen, L. (2004). Using easy optimization problems to solve hard ones.

Chapin, B. (2018). First Impressions – A Guide to Onboarding UX. Toptal Design Blog. <https://www.toptal.com/designers/product-design/guide-to-onboarding-ux>

Fronza, I. (2013). Cooperation wordle using pre-attentive processing techniques

Grant, R. (2018). *Data Visualization*. Taylor & Francis.

Herman, I., Melancon, G., & Marshall, M. S. (2000). Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1), 24–43. <https://doi.org/10.1109/2945.841119>

Kobourov S.G., Pupyrev S., Saket B. (2014) Are Crossings Important for Drawing Large Graphs?. In: Duncan C., Symvonis A. (eds) *Graph Drawing. GD 2014. Lecture Notes in Computer Science*, vol 8871. Springer, Berlin, Heidelberg. https://doi-org.ludwig.lub.lu.se/10.1007/978-3-662-45803-7_20

Drawing Large Graphs?. In: Duncan C., Symvonis A. (eds) *Graph Drawing. GD 2014. Lecture Notes in Computer Science*, vol 8871. Springer, Berlin, Heidelberg. https://doi-org.ludwig.lub.lu.se/10.1007/978-3-662-45803-7_20

<https://www.agilealliance.org/glossary/iterative-development/#author>. (2021, March 26). What is Iterative Development? | Agile Alliance. Agile Alliance | <https://www.agilealliance.org/glossary/iterative-development>

Hu, Y. (2012). Algorithms for Visualizing Large Networks. *Combinatorial Scientific Computing*, 525–549. <https://doi.org/10.1201/b11644-20>

Hughes, B., & Cotterell, M. (2009). *Software Project Management (5th Revised ed.)*. McGraw-Hill Education.

Hutchins, E. (2000). *Distributed Cognition*. San Diego IESBS University of California. - References - Scientific Research Publishing. (n.d.).

Kobourov, S. G. (2012, January 14). Spring Embedders and Force Directed Graph Drawing Algorithms.

Kobourov, S. G., Pupyrev, S., & Saket, B. (2014). Are Crossings Important for Drawing Large Graphs? *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, 234–245. https://doi.org/10.1007/978-3-662-45803-7_20

Manifesto for Agile Software Development. (2001). *Manifesto for Agile Software Development*. <https://agilemanifesto.org/>

Mazza, M. (2009). *Introduction to Information Visualization*.

McCarthy, J., & Wright, P. (2007). *Technology as Experience (The MIT Press)*. The MIT Press.

Meidiana, A. (2019). [PDF] *A Quality Metric for Symmetric Graph Drawings*

Mejtoft, T., Ristiniemi, C., Söderström, U., & Mårell-Olsson, E. (2019). User Experience Design and Digital Nudging in a Decision Making Process. *Humanizing Technology for a Sustainable Society*, 429. <https://doi.org/10.18690/978-961-286-280-0.23>

Norman, D. (2005). *Emotional Design: Why We Love (or Hate) Everyday Things (1st ed.)*, Chapter 1. Basic Books.

Norman, D. (2013). *The Design of Everyday Things: Revised and Expanded Edition* (Revised ed.). Basic Books.

Norman, D. A. (1993). *Things That Make Us Smart: Defending Human Attributes In The Age Of The Machine*.

Opila, J. (2019). Role of Visualization in a Knowledge Transfer Process. *Business Systems Research Journal*, 10(1), 164–179. <https://doi.org/10.2478/bsrj-2019-0012>

optimization. (n.d.). *The Merriam-Webster.Com Dictionary*. Retrieved March 26, 2021, from <https://www.merriam-webster.com/dictionary/optimization>

Oxford University Press (OUP). (n.d.). *learning*. *Lexico.Com*. <https://www.lexico.com/definition/learning>

Preece, J., Sharp, H., & Rogers, Y. (2015). *Interaction Design: Beyond Human-Computer Interaction* (4th ed.). Wiley.

Purchase H.C., Cohen R.F., James M. (1996) Validating graph drawing aesthetics. In: Brandenburg F.J. (eds) *Graph Drawing. GD 1995. Lecture Notes in Computer Science*, vol 1027. Springer, Berlin, Heidelberg. <https://doi-org.ludwig.lub.lu.se/10.1007/BFb0021827>

Purchase H. (1997) Which aesthetic has the greatest effect on human understanding?. In: DiBattista G. (eds) *Graph Drawing. GD 1997. Lecture Notes in Computer Science*, vol 1353. Springer, Berlin, Heidelberg. https://doi-org.ludwig.lub.lu.se/10.1007/3-540-63938-1_67

Rosenholtz, R., Li, Y., Jin, Z., Mansfield, J. (2010). Feature congestion: A measure of visual clutter. *Journal of Vision*, 6(6), 827.

Scrum Guide | Scrum Guides. (2020). *Scrum Guides*. <https://scrumguides.org/scrum-guide.html>

Shneiderman. (1983). Direct Manipulation: A Step Beyond Programming Languages. *Computer*, 16(8), 57–69. <https://doi.org/10.1109/mc.1983.1654471>

Shneiderman, B., & Aris, A. (2006). Network Visualization by Semantic Substrates. *IEEE Transactions on Visualization and Computer Graphics*, 12(5), 733–740.

Spritzer, A. S., & Freitas, C. M. D. S. (2008a). Navigation and Interaction in Graph Visualizations. *Revista de Informática Teórica e Aplicada*, 15(1), 111–136. <https://doi.org/10.22456/2175-2745.6015>

Spritzer, A. S., & Freitas, C. M. D. S. (2008b). Navigation and Interaction in Graph Visualizations. *Revista de Informática Teórica e Aplicada*, 15(1), 111–136. <https://doi.org/10.22456/2175-2745.6015>

Tufte, E. R., (2001). *The Visual Display of Quantitative Information* (2nd ed.). Graphics Press.

Teyseyre, A., Campo, M. (2009). An Overview of 3D Software Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(1), 87–105. <https://doi.org/10.1109/tvcg.2008.86>

The Definition of User Experience (UX). (n.d.). Nielsen Norman Group. <https://www.nngroup.com/articles/definition-user-experience/>

Wang, E. W. (2013). [PDF] A D 3 plug-in for automatic label placement using simulated annealing

Ware, C. (2013). *Information Visualization: Perception for Design (Interactive Technologies)* (3rd ed.). Morgan Kaufmann.

Ware, C., Purchase, H., Colpoys, L., & McGill, M. (2002). Cognitive Measurements of Graph Aesthetics. *Information Visualization*, 1(2), 103–110. <https://doi.org/10.1057/palgrave.ivs.9500013>

Chapter 8

Appendix A

8.1 Project plan and outcome

Project plan:

Week 1-2: Literature study

Week 3-4: Writing the theory section of the report

Week 5-7: Prototyping

Week 6-11: Implementation of the tool

Week 11- : Report writing

Outcome:

Week 1-2: Literature study

Week 3-4: Writing the theory section of the report

Week 5-7: Prototyping

Week 6-8: We familiarize ourselves with the library we are going to use for the visualization

Week 8-: Implementation of the tool

Week 11- : Report writing

Comments: We underestimated the process of choosing one and familiarize oneself with a visualization tool. Choosing one and getting to know the tool so that we could implement the graph took longer than expected. This set us back a bit on the time we had available for implementation. In the last 3 weeks we had planned for implementing we realized some of our designs had to be redone as they were not as good as we expected them to be. We had foreseen unexpected problems that would require our planning to change, and as such had a somewhat flexible time schedule where we had some time allotted for things such as this. The amount of time we would spent implementing was gravely underestimated however and instead of about 5 weeks we spent about 10 weeks on it. As such we lost time that we otherwise would have used for writing our thesis report. The cause for this is likely due to us wanting to improve the tool further and further, coming to the best possible solution we could. Since we came up with new insights and ideas all the time during implementation this went on for a long time. To avoid this harder requirements could have been set, then we would have had something to measure how finished we were with.

Chapter 9

Appendix B

9.1 Design questions to Andrew

1. Who are the end-users? What is their background knowledge?

The end-users will be engineers at Sinch, with a technical background, who will look at the blueprints. Their use-cases might be

- a) Using the blueprints to plan changes in a system, to tell which related systems might be affected by some significant change e.g. who is using this API I want to make big changes to? This is the dependency relation at the Interface level.
- b) The blueprints may be used by operational engineers for diagnostic troubleshooting purposes
- c) Engineering leadership team are interested to see how we are in control of the architecture and would like the opportunity to be able to self-serve on their architecture queries rather than having to ask a member of the Arch team. This is so they can maintain their situational awareness and be informed on the architectural topics they may be involved in decision-making around.

2. Are there any specific colors or symbols associated with different elements in the system already?

You get to choose some

3.a In which scenarios will the system be used?

I refer you to 1)

3.b What is the purpose of using the end-product? What are you hoping to achieve?

See also 1) The overarching objective is to catalogue to architecture and make it consumable to a wider audience (of engineers and engineering managers)

4. Are there any certain interactive functions you would like the end-product to have?

I would like to see ways of hiding complexity so the entire dataset does not have to be shown at the same time, and you can selectively pick parts you are interested in, so we can scale out the dataset to a high level of complexity whilst maintaining the ability to navigate through it without being overwhelmed by the complexity.

5. Does Sinch have any style guidelines we could take part of and use in our prototyping?

Not I am aware, but I would search the internet for best practises. I will look into this further for you.

6. How much information would you like to be available about an element before you have interacted with it? Do you have a preference on how this information is displayed?

I haven't developed any opinion on how much information is available, whatever works.

I was thinking there would be a web page where the majority of the page would show the object graph and there could be some section reserved for a toolbar where there might be some controls that determine what parts of the object graph are currently displayed. This is outlined in the doc I shared at the outset, I don't have any further constraints today apart from what is in the doc, I give you some freedom to try things and see what works!

7. Are there any other requirements you would like to have?

Not right now.

Chapter 10

Appendix C

10.1 Mid-fi prototype, iteration 1

Pictures on the first iteration of the mid-fi prototype that were not shown in the earlier chapters are shown here.

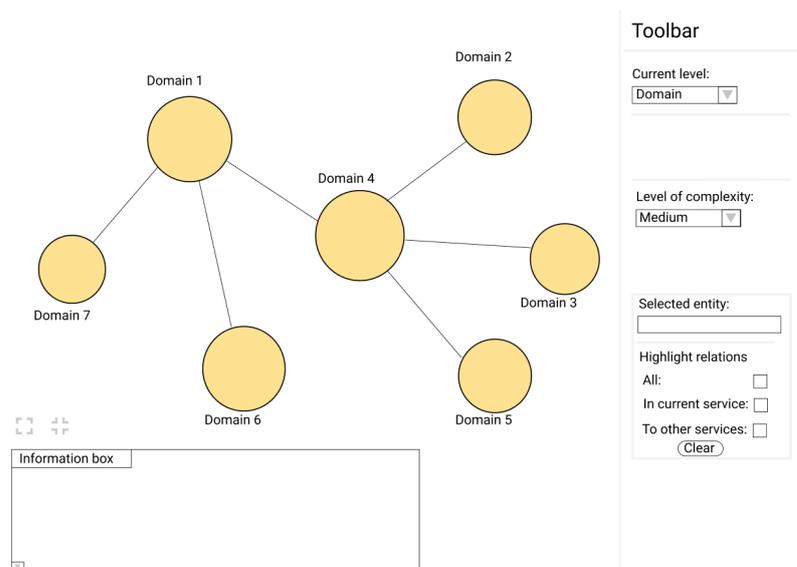


Figure 10.1: The domain nodes are shown. None of them are selected or expanded.

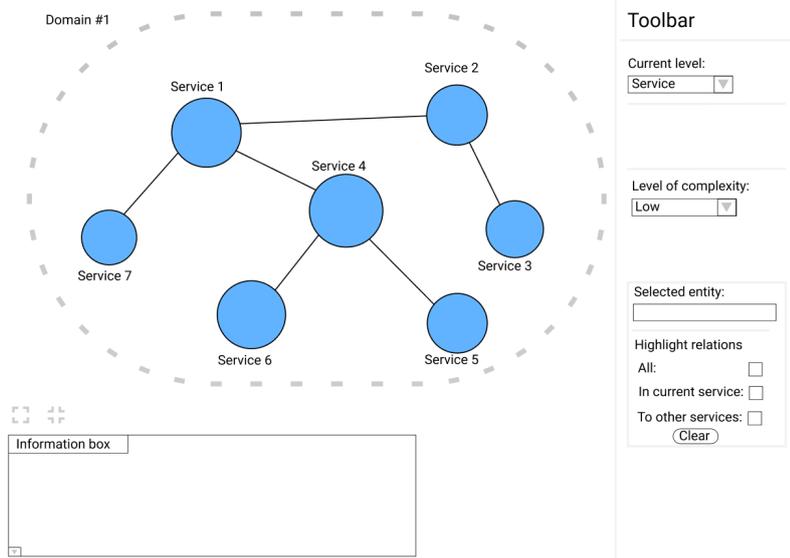


Figure 10.2: The service nodes that belong to Domain 1 are shown.

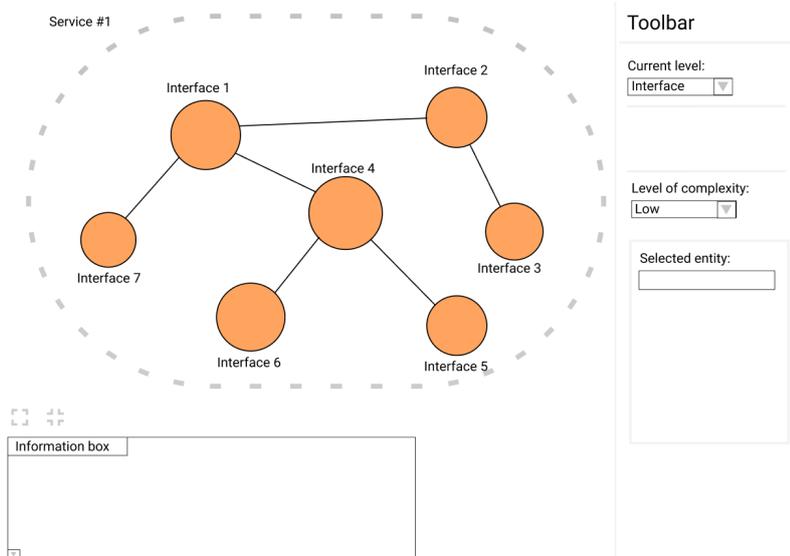


Figure 10.3: The interface nodes that belong to Service 1 are shown.

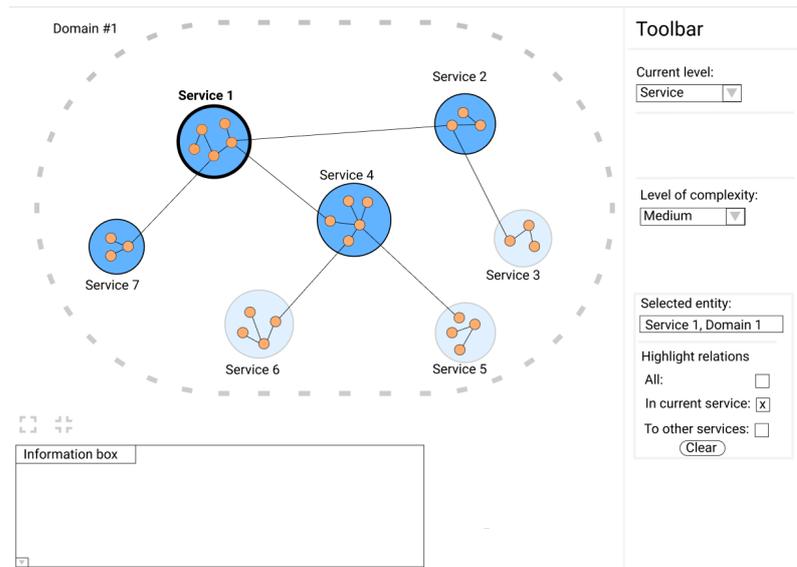


Figure 10.4: Service 1 within Domain 1 is selected. Highlighting is active.

10.2 Mid-fi prototype, iteration 2

Pictures on the second iteration of the mid-fi prototype that were not shown in the earlier chapters are shown here.

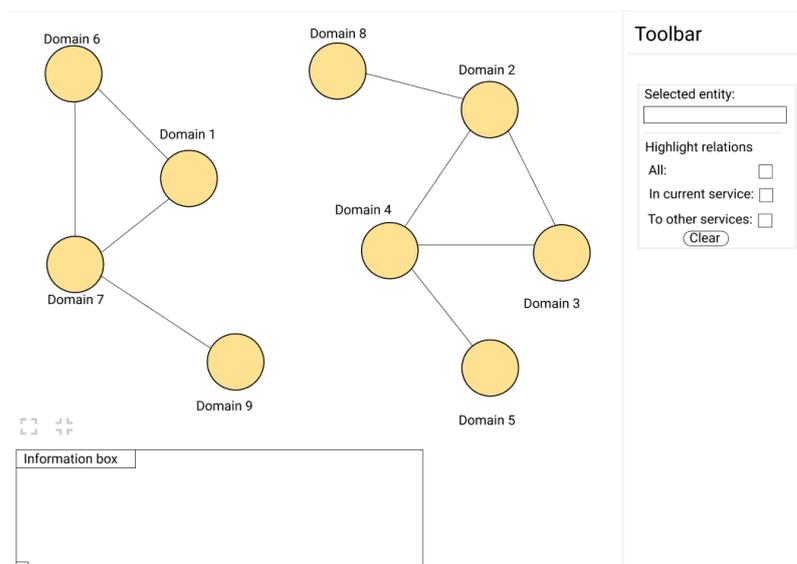


Figure 10.5: The domain nodes are shown. None of them are selected or expanded.

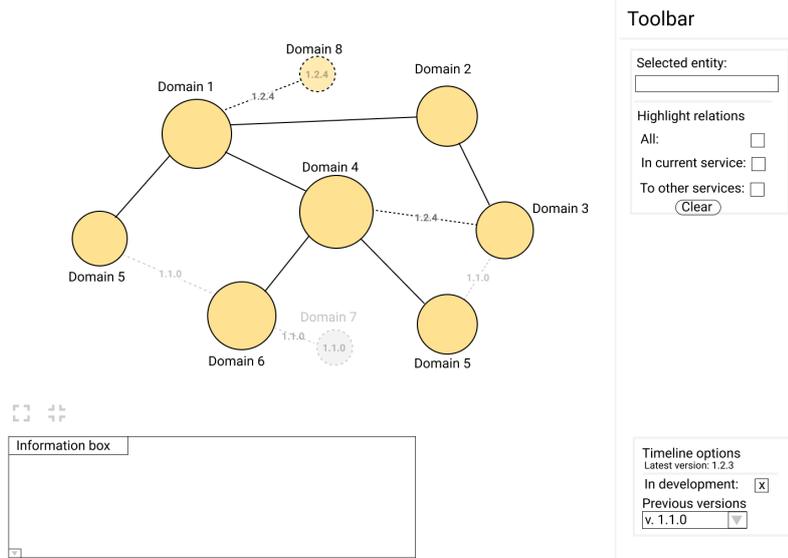


Figure 10.6: What is in development is seen with Domain 7. Domain 8 is a domain that has been there formerly. The dashed lines signify that a node or link is being developed. Dashed lines that are greyed out signify that the node or link existed in earlier versions. The numbers on links and nodes signify in which file version they are to be added or which version they existed in.

10.3 Final result

Pictures on the final result that were not included in the earlier chapters are shown here.

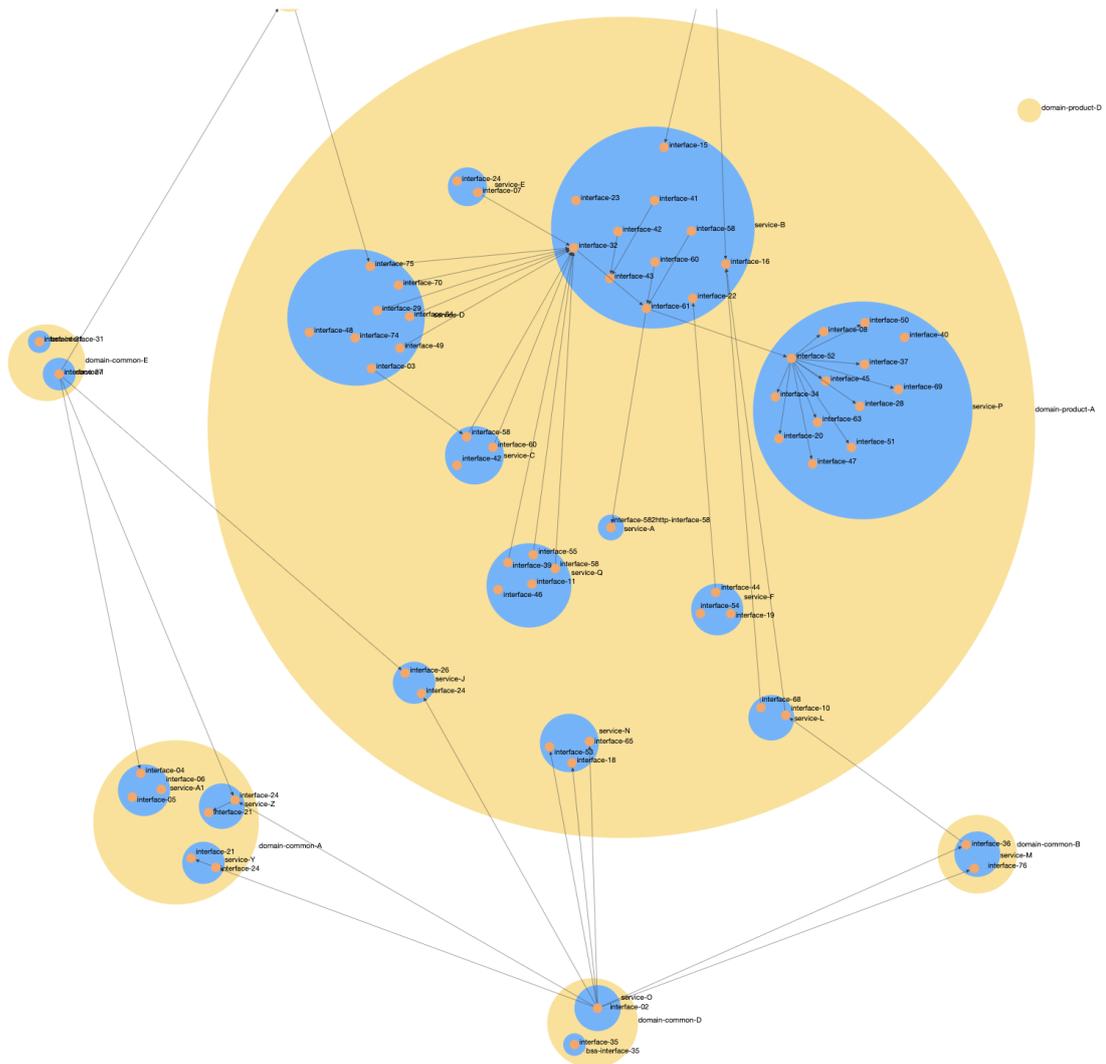
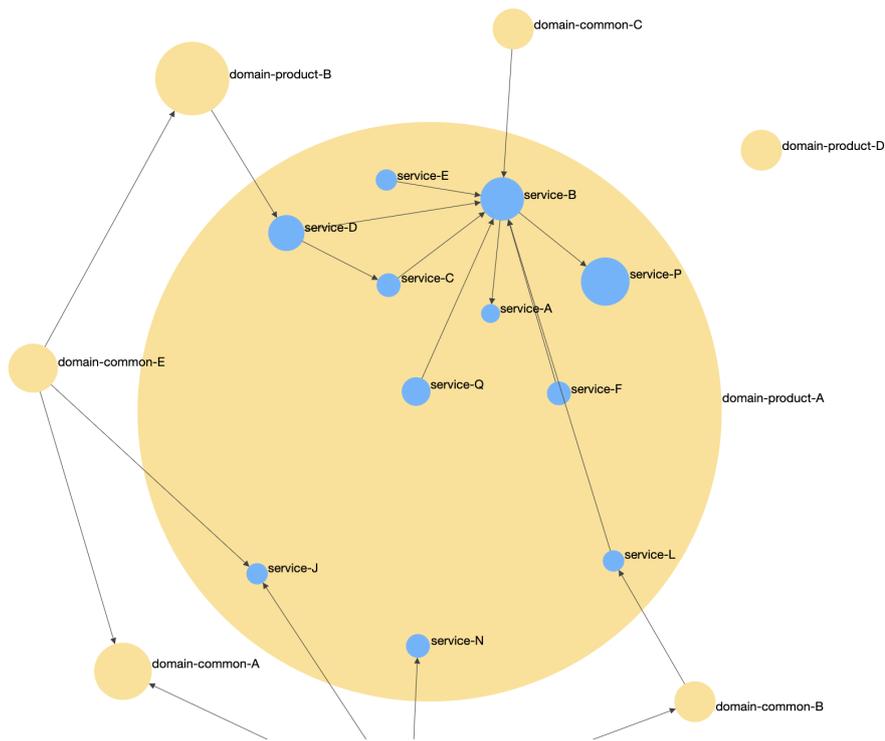
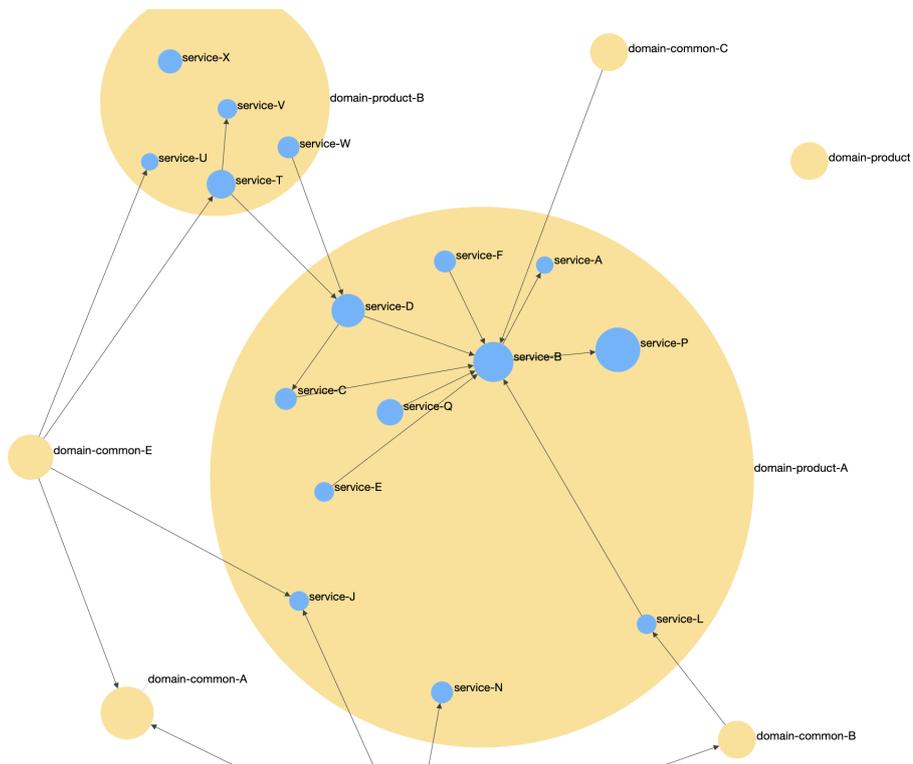


Figure 10.7: When all nodes are expanded the graph becomes very large and hard to comprehend.



(a) The user clicks domain-product-A, no other domain is clicked.



(b) The user clicks domain-product-A after domain-product-B already is clicked.

Figure 10.8: The service nodes within domain-product-A are placed differently in (a) and (b). This due to that domain-product-B already is opened in (b) when domain-product-A is clicked, leading to new forces affecting the domain-product-A's subnodes.