MASTER'S THESIS 2020

# Diagrammatic Programming Interface for Graph Traversal Algorithms


DEPARTMENT OF DESIGN SCIENCES
LTH | LUND UNIVERSITY

# Diagrammatic Programming Interface for Graph Traversal Algorithms

Oskar Rydengård and Daniel Andersson

# Diagrammatic Programming Interface for Graph Traversal Algorithms



*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)
Division: Ergonomics and Aerosol Technology
Supervisor: Joakim Eriksson
Co-supervisor: Anton Berghult
Examiner: Günter Alce

## Abstract

Graph databases are a modern and efficient way of dealing with large amounts of interrelated data. The company provides a software platform that collects, documents and visualizes data for a large variety of businesses. With a new major update in the works, the company proposed a new feature in the product which will be used by both in-house developers and customers alike. There is a need for a user-friendly interface for the design and construction of graph traversal algorithms for data retrieval.

Designing an interface that is intuitive and efficient for developers and end-users puts high demands on the design process to understand a variety of user groups. The main challenge consisted of abstracting domain specific terminology into a more general model. By applying a variety of user-centered design methods a solution was developed and evaluated. Using a node-based programming interface proved to be a viable approach for construction of graph traversal algorithms.



**Keywords**: graph database, interface, user-centered design, visual programming

## Sammanfattning

Grafdatabaser är ett modernt och effektivt sätt att hantera stora mängder av relaterad data. Företaget som detta arbetet utfördes hos erbjuder en mjukvarutjänst som samlar, dokumenterar och visualiserar data för en bred kundkrets. Med en kommande uppdatering har företaget förslag på en ny funktionalitet i produkten, som skulle komma att nyttjas av deras egna utvecklare såväl som kunder. Det finns ett behov för ett användarvänligt gränssnitt för konstruktionen av traverseringsalgoritmer för uthämtning av data.

Att designa ett gränssnitt som är intuitivt och effektivt för både utvecklare och slutanvändare sätter högra krav på designprocessen att tillhandahålla en förståelse för olika grupper av användare. Den primära utmaningen innefattade abstraktion av domänspecifik terminologi till en mer generell modell. En lösning utvecklades och evaluerades genom att applicera en samling av användarcentererade designmetoder.



**Nyckelord**: grafdatabas, gränssnitt, användarcentrerad design, visuell programmering, traveserings algoritmer

# Acknowledgements

First, we would like to thank the company that offered us the possibility to conduct this thesis work. We received much appreciated support and feedback from our supervisor and colleagues that made us feel at home.

We also want to express our appreciation for all the moral support from our family and close friends. We would not be able to do this without you.

Special thanks go out to our supervisor, Joakim Eriksson, for his great mentorship and guidance throughout this whole thesis work, as well as various courses throughout our studies at LTH.

Last, but not least, thank you Lars Welin, the photographer who gave us permission to use his beautiful shot featured on the cover of this thesis.

# Contents

# Terminology

**Application** Refers to the product that our implementation will be incorporated into.

**Field** An encapsulation of a graph traversal algorithm.

**Field Value** Data that is extracted using a Field.

**Resource type** Data categorization within the product. For example: user, computer.

**Step** A building block that relates to a particular operation in a graph traversal algorithm.

**Tag** Functional predecessor to a Field. Used in the current version of the product.

**Toolkit** Interface area that contains available components for construction of a field.

**Workspace** Interface area that contains selected components.

# Chapter 1
# Introduction

## 1.1 Background

Contemporary society is characterized by the growing interest in the ever increasing amounts of data stored about people, environment, infrastructures, financial information and metadata. This development has given rise to an interest in understanding how this data is interrelated. One example could be the case when a person is interested in seeing how many mutual friends he or she shares with another person. In later years, there has been a clear trend towards an increased usage of graph databases [1]. This type of database is emergent from the mathematical field of graph theory, which enables efficient handling of data, especially in cases where data interrelations between entities are of interest. With regards to performance, graph databases have an evident advantage when tasked with the ever increasing volumes of data, in comparison to relational databases. The host company (which desires to remain anonymous in this thesis) of this thesis provides a business-to-business, on-premises IT Asset Management software. It collects and stores large amounts of data about a company's IT assets, which includes, but not limited to users, hardware, databases, servers, licenses, certificates and other technical assets. This data is then presented in various views such as widgets, cards and tables. The software operates on a sophisticated, in-house developed graph database.

Currently, the company is developing the next major release of their main product. The company has performed a customer analysis and discovered that their user base is more diverse in their technical background than initially anticipated. This update is centered around better functionality, intuitive interface and streamlined workflow. A new feature for this release is what has been internally called a Field Builder. One of the main uses for it will be to allow users to create and customize *fields* that will retrieve requested information from the database based on a user's specification. A field is an encapsulation of a graph traversal algorithm. The company has assigned us with the task of investigating, developing and evaluating a prototype for an implementation of the Field Builder.

## 1.2 Purpose & Goals

This thesis will explore the possibilities of designing and implementing a graphical interface for creation of graph traversal algorithms. The purpose is to examine design factors that would facilitate the creation process based on a user's technical knowledge. A set of research goals which this thesis will address are defined as follows:

- Investigate a potential interface paradigm for implementation of fields
- Investigate suitable levels of abstraction for different types of users differentiated by their technical background
- Develop and evaluate a functional prototype of a suitable implementation of the Field Builder

## 1.3 Scope & Limitation

As the investigation and development of an implementation progressed, the scope of the thesis had to be narrowed. This decision was made as a consequence of the data gathered and analysed during various design methods that were applied in the second and third stages of the design process. The final prototype was aimed at primarily in-house developers with extensive knowledge and understanding of the underlying system on which the Field Builder operates.

We have applied a T-prototype approach, which entails focus on a particular aspect of the prototype. In our case we chose to focus primarily on the workflow of the building process of fields. Multiple related features that are relevant for the Field Builder are discussed to serve as a foundation for feature implementation. However, they were not part of the evaluation process as they are secondary in their value for this thesis work.

## 1.4 Related Work

Siau published a paper about the potential of visual object-relationship query language for user-database interaction [2]. He provides mathematical proof that the visual query language he created is relationally complete, meaning that it is at least as powerful in its capabilities as more traditional query languages such as SQL. His result indicates that a visual programming paradigm can be applied when designing a user interface for construction of database queries. This is a relevant result for this thesis as one of the goals is to investigate an interface paradigm that can be applied for implementation of graph traversal algorithms (or queries). While Siau's work covers specifically a language aimed towards relational databases, it is within the domain of database query languages, which are related to graph traversal algorithms.

Dahl and Lindquist has conducted research on the applicability of the visual programming paradigm as an interface between a program and user [3]. Particularly, they look into means to produce textual code through manipulation of images and gestures by a

user. They conclude that while there are difficulties in defining a general purpose visual programming language, they are feasible for implementation when the scope of application is more narrow. This is of relevance for our work, as a potential implementation of the Field Builder would interpret and translate a graphical configuration of elements into a textual query.

# Chapter 2
# Theoretical and Technical Background

In this chapter, we will present all necessary theoretical knowledge for the scope of this thesis. All of applied design methods will be covered in following sections of this chapter, and a brief introduction the domain of graph databases, visual programming, traversal algorithms and the application that this thesis builds upon.

## 2.1 Theory

In this thesis we chose to incorporate the user-centered design (UCD) methodology into our process. In the domain of usability and interaction design it is considered to increase usefulness and improve usability of the end product [4]. What makes user-centered design stand out from more traditional product design philosophies, is that it encourages the involvement of actual end users in the design process and focuses on what the users can, want and need, to use the product. It allows designers to better understand the end user's perspective and provides insights into the first time experience of the product, and it can give clues about factors such as the learning curve and potential problems with the design that complicates the interaction and usage of the product.

User-centered design is characterized by an agile and multi-phased process, usually consisting of five distinct phases. Every process phase consists out of several design methods that is chosen by the designers depending on the context and character of the project. The different phases focuses on the following:

- **Phase one**: Planning, scoping, and definition.
- **Phase two**: Exploration, synthesis, and design implications.
- **Phase three**: Concept generation and early prototype iteration.
- **Phase four**: Evaluation, refinement, and production.
- **Phase five**: Launch and monitor.

### 2.1.1 Concept Mapping

Concept mapping is a design method focused on creating a visual framework with the purpose of allowing new concepts to be created and added into existing domains of knowledge. Novak[5] points out that a concept map is never truly complete and should be revised multiple times, adding new concepts and creating new inter-connections. This framework could then be used as a foundation for further design processes and decision making regarding the development. Concept mapping process produces a design artifact in the form of a graphical tree-structure representing concepts and their interrelations.

### 2.1.2 Flowchart

Flowcharts is a exploratory method for designers which facilitates idea generation and gathering of new knowledge about the problem space [6]. They act as a foundation to challenge preconceived assumptions about the design and allow for theoretical experimentation with new sequences of interaction with the system. Flowcharts can be applied in different phases of the design process, making it a very flexible tool that evolves along with the product. In the initial, exploratory phases the flowcharts tend to be highly general, describing the interaction flow from a broad perspective. As the design process progresses

through the phases the flowchart has the potential to grow in level of detail, including new features and possible user actions.

Flowcharts are commonly arranged in a visual format, typically on a whiteboard with post-it notes or by using graphical software. Flowcharts describe sequences of actions and events in a process or scenario to achieve a particular user goal. Its main components are potential user actions, system responses to the action and decision points. The chart can branch into multiple paths, depending on the user input that the product encompasses. Related elements are positioned in close proximity to each other and are connected to each other by lines, similarly to a concept map.

A flowchart can also serve as a rough representation of the complexity level of the interface. A convoluted chart, with many decision points and highly interconnected components can hint at a potentially confusing user experience, either of the interface as a whole or a particular aspect. Similarly, it can hint at a lack of detail, exposing potential gaps and hinting at areas requiring attention.

Upon completion this method produces a graphical artifact representing an overview of the flow of user decisions and actions. It is then used as a guideline in early stages of prototyping for creation of basic elements which would represent the main interaction flow of the system. Once a prototype is complete, it should provide the possibility of traversing the flowchart in an intuitive way, following each available path from a starting to an end point. Any issue or difficulty that arises during the testing of the prototype points towards either a potential design flaw in the prototype or an overlooked functional gap in the flow chart.

### 2.1.3 Data-driven User Analytics

Data-driven user analytics is a research method for gaining information through analysis of user data and metadata. It can provide an insight into user behaviour and their needs [7]. For example, popular search terms can give a clue about frequency of use of features or features that are hard to navigate to. This can then be used as valuable data to aid further design decisions, either to add new features or modify existing ones.

User analytics has high synergy with other design methods like task analysis, personas, scenarios and interviews. This method can reveal users that frequently use the system or some specific feature, which is useful for finding suitable people for conducting interviews, as those people are more likely to yield more useful feedback due to having more experience. This shortens the user selection phase considerably, allowing us to get in contact with relevant users directly.

Depending on the sophistication of the analytics tool the user's data can be differentiated between different user types. For example users could have a certain role assigned to them, such as system technician, IT manager or operations manager. Such information can be useful to gain a better understanding of different usage patterns. Additionally, this information is very useful for creation of personas and scenarios.

### 2.1.4 Card Sorting

Card sorting is a method used to facilitate the process of grouping content and features for navigation and organization in the user interface[8]. To begin with, a selection of terms related to the product is compiled. These can be gathered from previous design methods such as concept mapping and flowcharts. Then, each relevant term is written on paper cards, which are then presented to users as either a shuffled deck or spread out in random fashion on a table. Users are then asked to sort the cards into different piles corresponding to their similarity in the context of their mental model.

Next, the users are asked to place the sorted piles into larger groups that seem to belong together and to come up with a fitting label for each group. Created labels are then written on a post-it note and placed above each group. Group labels and their relations are then analysed and used as guidelines in following stages of the design process. Nielsen mentions a situation where different users use different labels for similar groups, due to the verbal disagreement phenomenon[8]. This phenomenon describes the team resorting to subjective judgement to decide upon a suitable name for the groups used in future iterations, using the names produced by the users as inspiration. Consequently this can result in different names being used to describe the same concept.

A useful aspect of this method is that it is compatible with the think-aloud protocol, allowing testers to listen in on a user's thought process as they advance through the process. This greatly aids with an insight into user's mental models regarding subject at hand. Additionally, it can highlight terminology that is likely to be misunderstood by the users.

### 2.1.5 Scenarios

Scenarios are a useful method that allows designers and developers to describe a meaningful narrative that creates a context around the product[9]. A scenario is preferably written from a persona's point of view, with each relevant persona getting at least one scenario each. A test person can be seen as an embodiment of a persona, and should be selected accordingly to match some of the key-traits of the persona, for example skill level and background. Scenarios can be applied as a foundation for the test instructions in later design phases, for example in prototype evaluation.

Written in a story format, a scenario typically includes a goal from a users point of view, how he or she goes about achieving that goal and finally the result. It can be written describing *how* something happens in a step-by-step sequence, describing which actions user needs to perform to progress towards the goal. Additionally they can also describe *what* happens, from a higher level perspective, focusing on the user experience itself. Viewing a particular task from a user's perspective can be a great way to facilitate discussion about user flows, features and UI design in the team.

### 2.1.6 Interviews

One of the most common ways of involving a user into the design process is by conducting an interview. Interviews are a method for extracting exploratory information about the problem space and the users in early stages of the design process. These can be organized in different ways, with two main ones being structured and unstructured [10]. Structured interviews feature a predefined script of questions, while unstructured are more free, allowing for an open discussion about a topic. A semi-structured interview can be applied through a combination of these two methods. This format entails having pre-constructed questions, followed by some open-ended, baseline questions that create opportunities for the interview to branch into new topics or dive deeper into a predefined one.

### 2.1.7 Task Analysis

Task analysis a method applied for gaining an understanding of how a user's work flow in a current system is structured. Data for task analysis can be collected through interviews, questionnaires or by observing users interacting with a system directly. A task, in context of this analysis, refers to a broad definition of actions or mental processes involved in achieving a particular goal. Aspects such as available options at a particular stage in a sequence of actions, decision points, common mistakes, user inputs and system responses/outputs are the focal point of this method.

This method is a complementary way of processing and extrapolating data from other design methods applied during the design the process. This plays an important role in following design methods, as it facilitates the understanding of a user's mental model of an existing system. According to Kuniavskiy [10], task analysis is appropriate when the problem domain is known and the goal is to understand how users are currently working within that domain.

### 2.1.8 Competitive Analysis

Competitive analysis is a method where the purpose is to evaluate related products or services available on the market. A valuable aspect of this method is that it allows for the assessment of a product from an end-user's point of view [10]. A suggested approach is to define a scenario and attempt to follow it through in different products, taking note of factors such as usability, learnability, feedback and visual design. In creation of such a scenario a useful starting point is a definition of a desirable end goal for a potential user of one's own product. This same scenario should then be applied during later phases of the design process to evaluate and compare those same factors mentioned. By evaluating key differences opportunities for differentiation can be found and used to gain insight and competitive advantage. It is important to take note of differences as well as similarities between products.

### 2.1.9 Prototyping

Prototyping is the process of realization of some aspect of a product or interface concept that is under development. It is an opportunity to translate insights and efforts from previ-

ous design methods into a more tangible form. Prototyping is generally differentiated by level of fidelity in comparison to the final product or a vision of it.

Low-fidelity or Lo-Fi prototyping is done in the early stages of a project, mainly due to the cost efficiency and ease of implementation [11]. A typical approach to this level of prototyping is so called paper prototyping, which consists of paper cut-outs representing various user interface elements. The simplicity of this approach allows for quick iterations, mock-ups of ideas to arrive at a viable starting point for a higher fidelity prototype.

High Fidelity or Hi-Fi prototyping is a more refined, functional implementation of a prototype. For software products this typically involves coding an interactive interface that has most of the main functionality represented.

### 2.1.10 Personas

A persona is an artificial representation of a group of users who share similar aspects, qualities and goals [12]. The purpose of a persona is to conceptualize a user, as a reminder about whom a product or a service is aimed at. This method helps designers to adapt the design process in a effective way, saving time and resources. Personas can, and preferably should, be extrapolated from other design methods that gather data - such as interviews, task analysis and competitive analysis. A constructed persona is a way to consolidate data about users in a summarised format. Further, they can be used as a pragmatic centerpiece for creation of authentic scenarios with high fidelity.

### 2.1.11 Heuristic Evaluation

Heuristic evaluation is an informal usability inspection method that enlists team members of varied expertise levels and areas to asses an interface against a set of agreed upon best practices in the industry and/or the company [13]. The purpose of this method is thus more about revealing critical flaws, rather then providing insight about novel design opportunities. This method is conducted internally, before actual users are involved for further evaluation of an interface, making it cost-effective, as it offers a way to conduct usability testing in the earliest stages of prototyping. This creates space for detecting faults and deficiencies early, making later testing phases more efficient, by opening up for new opportunities for actual users to give feedback about.

Further, studies have shown that combining multiple inspections methods, such as cognitive walkthroughs, can lead to better results, as some problems that are overlooked by inspection are able to be found through user-testing and vice versa.

Preferably each participant should evaluate an interface individually, and then comparing the results with others. As with competitive analysis, which is focused on third-party products, both positive and negative aspects should be taken into account.

## 2.2 Theory

### 2.2.1 Graph Databases

A graph database is a more recent development in the database domain and is rising rapidly in popularity compared to the more traditional relational database model [14]. A graph database is based on graph theory from mathematics which is used as a foundation for the data structure on which the database operates. The main advantages of graph databases are their speed and flexibility when handling large amounts of interrelated data, making them particularly useful for ever growing data sets and increasing number of edges between vertices.

A graph is composed of two types of elements, *vertices* and *edges*. A vertex represents some kind of entity like *Student*, *Course* or *University*. An edge is a relationship between two vertices that describes the nature of their association to each other. For example *ENROLLED* can be a edge connecting vertices of type *Student* and *Course* with each other. Both vertices and edges can have a number of properties describing information related to them. Using the same examples, *Student* can have the properties of *name*, *studentID* and *socialSecurityNumber*. See figure 2.1 for a visual representation of graph structure.

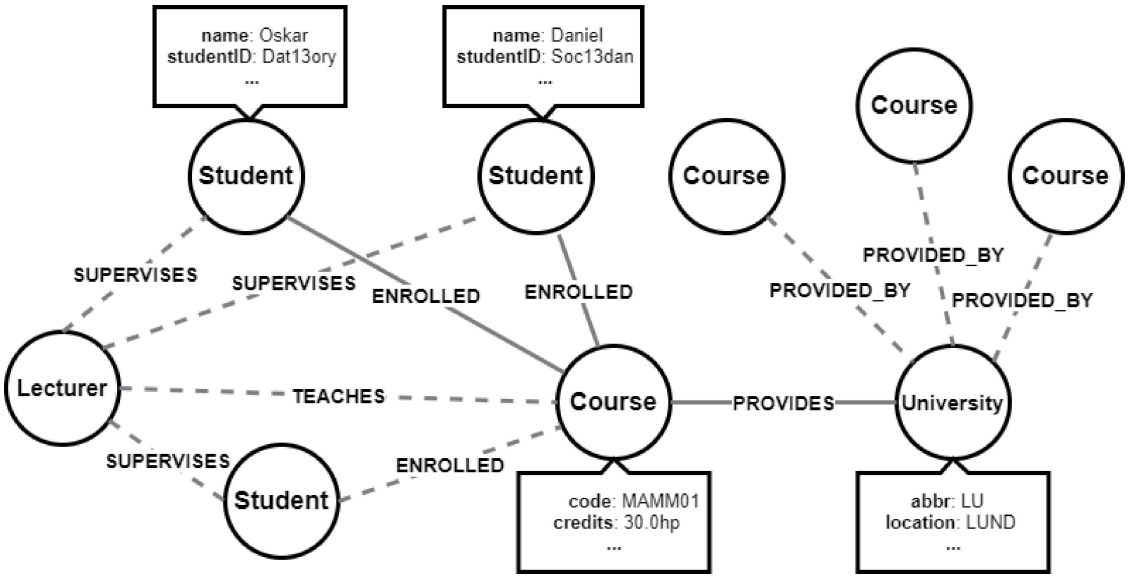



**Figure 2.1:** Example of a graph database, the edges are often directed in graph databases but in this example we have chosen to not emphasize that detail.

### 2.2.2 Traversal Algorithms

A traversal algorithm is a computational process of visiting nodes in a graph structure [15]. Although the original definition defines graph traversal as a process for visiting each and every node in a graph, this is not strictly applicable in the context of this work and deserves some clarification. Graph traversal is originally a term inherent to graph theory

and not graph databases. Consequently, one of the main challenges in database design is optimizing for the retrieval speed of information stored on request. Traversing each and every node on every request would be very resource intensive and ineffective, thus certain techniques such as caching are utilized to reduce processing requirements.

A database request is performed with certain constraints defined by the query. For example the query can request certain properties such as name, type or relationship to other nodes. Once the query is processed the requested data that fulfills the defined constraints is returned.

A potential query for the graph illustrated in figure 2.1 could be the following: Return the course code for a course provided by a university with the abbrevation `LU`, where both a student with the student id `Dat13ory` and `Soc13dan` are enrolled. The traversed edges are illustrated by solid lines.

### 2.2.3 Visual Programming

Visual programming is a programming paradigm that revolves around the ability to create programs through the usage graphical elements instead of writing traditional text based code [16]. This is often achieved by assigning semantic context to a predefined group of graphical elements that represent common programming concepts such as variables, loops, conditional statements and logical operators. Usually a metaphor of "boxes and arrows" is applied when describing this paradigm, as a common application of it involves graphic elements arranged and connected in some kind of sequential manner.

One potential advantage of visual programming languages is the ability to provide a quick and broad overview of a program's structure or some part of it. Particularly a representation of the data flow can be illustrated through the usage of directional arrows between various graphic elements depicting logical components of the program. A logical component can have multiple inputs as well as propagate its output towards multiple other components. Visual programming is a paradigm with research and applications dating back to the 80s[17]. As a result of a general trend in software applications to provide interfaces with a higher abstraction level, visual programming has seen an increase in adaptation in various fields spanning from educational resources such as Scratch [18] and Kodu [19] to industry specific applications such as Unreal Engine [20], NODE-RED [21] and Blender [26].

## 2.3 Description of application context

### 2.3.1 Purpose and technical background

The application this thesis revolves around is a type of Content Management Database (CMDB) and IT Software Asset Management (ITSAM) system for IT-infrastructures. It offers insights and documentation regarding an organizations internal IT-environment, both on premise and cloud based artifacts. This includes everything from servers, application, virtual machines, server clusters and certificates to user domain accounts and physical network switches and gateways. In other terms, this application can be seen as a search engine for the internal IT-infrastructure within an organization. A typical use

case could be that a system technician wants to know on which physical machines a service is hosted at, which normally requires a lot of scripting and knowledge of how the infrastructure is structured to accomplish.

The application inventories the IT-environment on daily basis, using integrations to common data sources such as Microsoft Azure[24], Amazon Web Services[23] and Active Directory[22] and then models the data in an in-house developed revision based graph database, optimized for this purpose and the features that the application offers. It is not uncommon for the same entity to appear in multiple data sources. For example the same asset can get discovered from different integrations and therefore exist in multiple retrieved sub-graphs from different integrations. After a snapshot of the IT-environment have taken place, the application "stitches" the model, meaning that the different sub-graph from individual integrations gets merged into a single unified graph network. The application uses sophisticated algorithms to identify these assets in order to link the sub-graphs together by merging the same assets together and unify the graphs. Once the stitching is done, a complete map of the IT-environment that bridges the different domains and application environments within the organization is created.

### 2.3.2 Data presentation and categorization

The application offers several different views for data presentation, these are:

- **Table explorer**: This view consists of a spreadsheet, where the rows represents an entity within the IT-environment and the columns shows values extracted from the selected tag. Cells can contain multiple values depending on the output from a tag. See figure A.1 in appendix.
- **Properties**: This view shows only one entity and the most relevant tags for that type of resource to provide a quick overview. See figure A.2 in appendix.
- **Dashboard & Widgets**: The dashboard view consists of multiple widgets that shows KPIs (Key Performance Indicators), which are extracted form underlying tags. This offers a quick overview of the most relevant metrics for the context of the dashboard. Multiple predefined dashboard exist, for example Active Directory overview and Azure license overview. See figure A.3 in appendix.

All assets that exist in the application are divided into different resource types depending on the nature of their data. A resource type could be file systems, virtual machines or certificates, a complete list is presented in the appendix figure A.4. They play a vital role when it comes to the creation of tags, because the resource type is often used as an constraint in the traversal algorithm. By applying smart caching, the traversal algorithm does not need to visit all nodes in the graph network, increasing performance remarkably. For readers familiar with SQL, the concept is similar to the *FROM* statement, in terms of reducing the scope of the extracted data.

### 2.3.3 Tags

This, and the two following sections covering fields and relationships describe the technical implementation of the underlying structure regarding fields. To retrieve data from

the graph database and present it, the application uses *tags*. A tag consists of a traversal algorithm and some metadata. The traversal algorithm can be seen as an instruction set that specifies how the application finds the requested value in the graph database, which sometimes contains millions of nodes and edges. The values can be classified as static or dynamic. Static values are defined by user input via tags and will not be modified by the application, and dynamic values can not be modified by users and strictly contain inventoried values. The tags are created either by developers at the company or by customers. A set of default tags is bundled with the product. Tags created by the developers can be potentially more powerful and flexible than what a customer can create, because tags created by developers are implemented using *Gremlin*[25], which is a query language specifically developed for implementation graph traversals. On the other hand, when users create tags, they do it through the application interface and are limited to only three types of tags:

- **Fixed value tags**: These tags contain non-inventoried static values. Often defined by customers.
- **Value from another resource**: These tags imports values from another resource type. They act as a bridge between different resource type and allows values from one context to be displayed in another.
- **Value from another tag**: These tags copy values from other tags.

A feature available for these tags are filters. This could for example be a regular expression based filter that only allows values that matches that regular expression to be shown in the table explorer. Additional features for non-fixed values tags are arithmetic functions such as sum, average, count and unique.

The application is currently going through a major rework for an upcoming release, and the current tag functionality is being replaced with *fields*. Fields share a lot of similarities with tags but are based upon predefined blocks of Gremlin code, internally called *Steps*. These blocks will be presented in the following subsection named fields. Fields have additional functionality irrelevant for the scope of this thesis. Overall, fields are a more flexible and powerful version of tags.

### 2.3.4 Fields

As previously stated the tags are currently being reworked to be become more powerful and support additional features. To avoid the perception among users that tags are only values attached to assets, and to align tags with the new product vision, the name was changed to fields. In contrast to using Gremlin as a way of representing the traversal algorithm to extract data from the database, fields use an new in-house developed query language that adds a layer of abstraction on top of the underlying gremlin language. The operations and features of this query language is called steps and play a fundamental role in this thesis work, since the interface that will be developed will convert a graphical schema into this specific language. Therefore a brief introduction of the technical implementation of fields will be presented together with the syntax of the language. An example of a field implementation in pseudocode can be seen in figure 2.2.

```
FieldConfigString {
        id: "ipIpAddress"
        name: "IP IP Addresses"
        description: "IP addresses on IP vertex"
        cardinality: SET
        fieldType: USER
        fieldFormat: StringFieldFormat
        constraint: TraversalFunction<Vertex>(
           ↪ ElementTypesConstraint("ipaddress"))
        rules: [
                FieldRule: {
                        id: ipv4
                        stopRule: false
                        constraint: TraversalFunction<Vertex
                           ↪ >(identityConstraint)
                        function: TraversalFunction<
                           ↪ FieldValue>(identityConstraint).
                           ↪ VertexPropertyStep("iPv4address"
                           ↪ )
                },
                FieldRule: {
                        id: ipv6
                        stopRule: false
                        constraint: TraversalFunction<Vertex
                           ↪ >(IdentityConstraint)
                        function: TraversalFunction<
                           ↪ FieldValue>(IdentityConstraint).
                           ↪ VertexPropertyStep("iPv6address"
                           ↪ )
                }
        ]
}
```

**Figure 2.2:** Pseudocode example of a field implementation

The attributes `id`, `name`, `description` and `fieldType` are metadata for the field. The `fieldType` attribute describes whether the field was created by a user or if it came pre-bundled with the application.

The attribute `cardinality` can be set to SINGLE, SET and LIST and defines how many values the field will return. LIST represents multiple values as output and SET represents multiple unique values.

`fieldFormat` specifies the format of the returned values and can be set to all primitive data types and some custom made formats, for example `IP addresses` and `Currency`.

The `constraint` attribute specifies a resource type within the product, used as an constraint for which vertices the traversal algorithm will be applied on. In the example code, the constraint is set to only traverse paths in the graph that starts on a vertex of the type `ipaddress`. An `IdentityConstraint` translates to "no constraint" applied, meaning that every value passes through.

The attribute `rules` is a list containing `FieldRules`, every individual rule specifies a traversal in the `function` attribute to extract values from the graph with an additional constraint that is a subset of the general constraint specified on the field configuration level, thus decreasing the scope and optimizing the traversal further. The `stopRule` attribute specifies whether the traversal should skip the following rules if a value is found for that particular rule.

`TraversalFunction<?>` is a class that represents the traversal algorithm, in other words the graph walk. A Traversal function got a generic parameter for the output type, which can be:

1. **`Vertex`** Some traversals return a vertex, for example the constraints that filter out the vertices of a specific type in the graph or `Relationships` that will be described in detail in the next section.
2. **`Field`** A traversal function can also return a field.
3. **`FieldValue`** A traversal function can return values referred as field values.

`TraversalFunctions` are defined by a combination of `Steps`, that together form a query. `Steps` can be seen as a operation that will be executed at a particular position in the graph walk. The example function in the ipv4 rule utilizes a `VertexPropertyStep`, which fetches the values from the attribute `iPv4adress` from all the vertices of type `ipaddress` in the scope of the traversal function. However, in this example traversal function is highly simplified for demonstrational purposes. Most traversal functions include several steps.

A wide range of steps is available for usage and new steps are continuously being implemented whenever the need to solve new use cases arises. Therefore only a few steps will be covered in this paper to get a feel for the capabilities of the steps. There exists some steps that are arithmetic in their nature, for example the `FieldValueAggreationStep`, which can calculate the number, sum or average of a collection of extracted field values. And `VertexFieldValueMathStep` that can perform the traditional mathematical operations on a collection of field values.

Some steps are used to filter out a subset of the set of vertices defined by the previous steps in the traversal function, for example `VertexFilterStep`, which has an traversal function as an input parameter that act as a filter. Other steps are used to navigate in the graph, for example the `VertexEdgeToVertexStep`, which performs a graph walk from one vertex to another specified by the direction and name of the connected edges originating from the vertex that the step is applied on.

At last, there are some steps that are closely related to the properties of a vertex, for example `VertexPropertyStep` as earlier mentioned,`VertexFieldValueStep` and `VertexHas`. These steps are mainly used to extract properties or the state of a specific vertex.

### 2.3.5 Relationships

As an alternative to define every traversal function completely made out of steps, relationships can be used as a shortcut to perform navigation within the graph. Relationships are

basically a series of steps and act as an abstraction when creating fields. By using relationships the creation of fields becomes more simplified because it requires less knowledge of how the graph is structured internally e.g. vertex types and edge names to navigate from vertex A to vertex B. It also increases the scalability of the program avoiding "shotgun surgery" when making changes to the graph structure or refactoring multiple fields.

```
Relationship {
        id: "associatedIp"
        description: "IP␣addresses␣associated␣to␣a␣system"
        outConstraint: ElementTypesConstraint("
           ↪ computersystem")
        inConstraint: ElementTypesConstraint("ipaddress")
        outTraversal: TraversalFunction<Vertex>(
           ↪ identityConstraint).VertexEdgeToVertex(OUT, "
           ↪ HostedAccessPoint").VertexEdgeToVertex(OUT, "
           ↪ BindsToLANEndpoint")
        inTraversal: TraversalFunction<Vertex>(
           ↪ identityConstraint).VertexEdgeToVertex(IN, "
           ↪ BindsToLANEndpoint").VertexEdgeToVertex(IN, "
           ↪ HostedAccessPoint")
}
```

The pseudocode above shows how a typical relationship can be implemented. The relationship contains two attributes `id` and `description`, that contains a unique id that is used internally in the system and an description of the relationship. This is followed by two constraint attributes `outConstraint` and `inConstraint` that specifies the vertex types the relationship can be applied on. In the example above the relationship represents paths in the graph originating from a vertex of type *computersystem* to a vertex of type *ipadress*. Following the constraint attributes are two traversal functions `outTraversal` and `inTraversal`, just like traversal functions in fields they are made out of steps. The reason for relationships to have both an in and out traversal is that you sometimes have to take different paths in the graph to navigate from vertex A to vertex B or vice-versa.

As seen in the example, the traversals utilizes a `VertexEdgeToVertexStep` described in the previous fields section. If we interpret the *outTraversal* attribute we can see that it starts on a vertex of type *computersystem* specified by the *identityConstraint* that get its constraints from the *outConstraint* attribute. Then, it checks if any outgoing edge exists from the vertex with the name `HostedAccessPoint`, if that is the case the traversal function continues and follows that edge to a new vertex. Then the same procedure repeats. At the new vertex it checks if that particular vertex has an outgoing edge with the name `BindsToLANEndpoint`, if that is also the case the traversal function lands on a vertex of type ipadress and the graphwalk is completed. In this particular example the two traversal functions are mirrored.

## 2.4 Prototyping tools

### 2.4.1 Adobe XD

XD is a UI/UX design tool developed by Adobe [27]. It is the software of choice at the host company for creating digital prototypes of the interface for the upcoming version. All elements used such as buttons, icons, input fields, colors are available within design guidelines created in XD.

### 2.4.2 JavaScript React

JavaScript React is an open-source library for building front end interfaces, developed by Facebook and is widely accepted in the industry as one of the most sophisticated, supported and flexible libraries available on the market [28]. Based on these qualities as well as React currently being adopted in the company, we have opted for using it as our platform for the implementation.

# Chapter 3
# Design Process

In this section outcomes and insights from all of the applied design methods will be presented. While the theory for user-centered design process describes five distinct phases, the last phase focusing on *launch and monitor* is discarded as this project has a fixed lifespan which does not include a proper release. Monitoring and release presumes that the project has a phase which includes deployment and continuous analysis and design improvements during it's life cycle.

As a starting point we conducted a literature study to familiarize ourselves with central topics of this thesis - graph databases, traversal algorithms and visual programming. Then we applied multiple user-centered design methods to gain an understanding of users needs and mental models of the current product. Consequently, a Lo-Fi, followed by a Hi-Fi prototype of the interface was developed, tested and evaluated.

## 3.1 Method selection

We defined our process by first studying 100 commonly used design methods from a literature compilation named *Universal Methods of Design* [29]. Then, we graded all of the methods based on how suitable they were in the context of this thesis. Afterwards we selected around thirty of the methods with the highest score and began thinking of the synergy between the methods. Some methods offered high synergy between each other, which allowed for more effective use of the related methods. These methods could act as input and preparatory work for methods used at later stages of the design process. For example design methods Personas and Scenarios have a natural synergy, with the former method providing data to the latter. Out of 30 candidates we selected 11 methods that provided strong synergy with each other. An illustration over the chosen design methods can be seen in below.

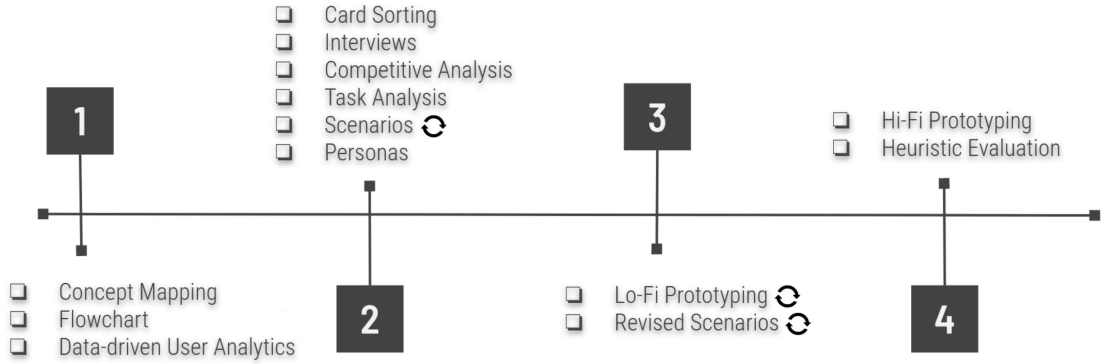



**Figure 3.1:** Design Process

## 3.2 Phase one

In the first phase of the design process, we focused on increasing our knowledge in the domain of visual programming, design methods and graph databases by studying relevant research papers, presented in related work section. We also spent time to get acquainted with the application in it's current state, to better understand its features, workflows and use cases.

### 3.2.1 Concept Mapping

We found this process to be applicable for this project, as we identified two major, distinct domains of knowledge - graph databases and interaction design, which we had to consolidate into one unified framework. The goal with this method was to facilitate a better overview and understanding of interrelations between concepts of the two main domains and how they are related to each other in the context of our project.

As a starting point we reflected over the main goal and scope of this thesis work. This led us to formulating the following focus area topic - "Facilitation of information retrieval" as a general root from which we could branch out to related concepts. In addition, we wrote down a list consisting out of 15 terms that we believed were the most important concepts related to our problem domain. These terms were then written on post-it notes and arranged in a tree-like structure on a whiteboard, with more general concepts at the top, getting more specific towards the bottom.

The first concept map resulted in a "string map", which is a map with one isolated branch without any cross-links and only a few propositions. Cross-links are relationships between concepts in different domains of the concept map. This branch consisted mainly of technological terms related only to the domain of graph based databases. We experienced difficulties connecting concepts from that domain to the domain of interaction design. In the literature, a string map is described as a sign of poor understanding of the concepts and their relations [5]. While this result appeared unimpressive at first, it was not completely unexpected, as this was the first method we applied. After further reflection we recognized that our initial focus question was too narrow in scope, forcing our concepts to be focused primarily on the technical aspects of the problem.

To widen the scope of the focus question topic we redefined it as "Increase usability", which opened up a new path in the map focusing on "how?" of the design process, complementing the previously single path which focused on the question of "why?". This offered a natural flow into concepts such as graphic user interfaces, complexity, discoverability, feedback and more. Consequently this presented an opportunity to create cross-links between the two domains.

As a means to create cross-links we defined several new concepts described through metaphors. This allowed us to create meaningful cross-links between the two domains and progress further down the map towards more specific concepts. This was aided by a rough, preliminary sketch of the interface which contained main graphical elements grouped by functionality. This step gave us an incentive to assign labels to graphical elements to facilitate the discussion around related concepts. In addition, labeling the elements helped us define new concepts and make new connections between previously separated domains.

Creation of propositions was achieved through splitting some of the existing concepts into multiple new representations viewed from different perspectives. For example a “Field Network Diagram” is a user friendly visual representation of the technical concept “Traversal algorithm”, thus allowing us to easier define linking words that acted as cross-links between the domains.

Following each path of the map and reading out loud the concepts as well as the linking words resulted in a meaningful narrative, which gave a sense of completeness and cohesion between the domains.

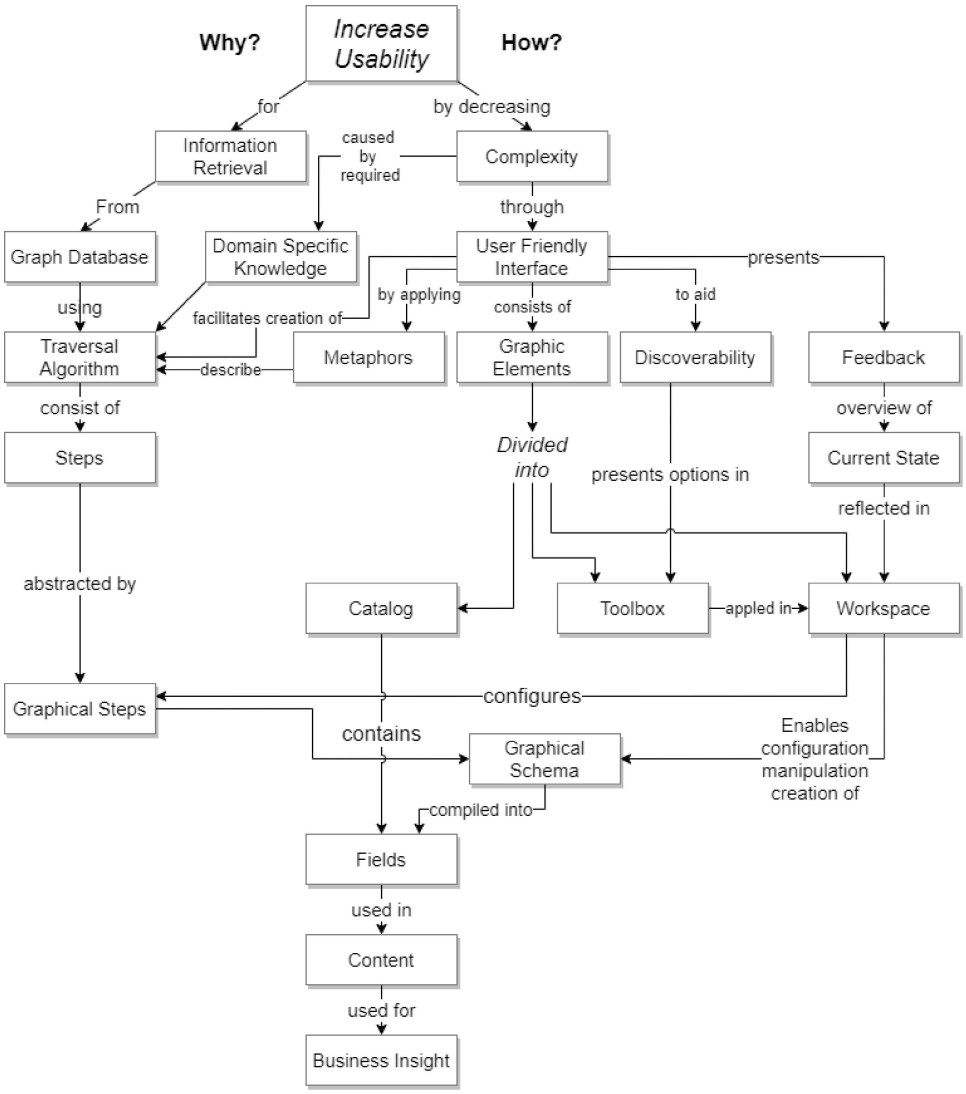



**Figure 3.2:** Concept map constructed during the first design phase

### 3.2.2 Flowchart

The outcome from this method consisted of a bare-bones chart, only focusing on the most essential interaction sequences. We chose to leave the interaction flow regarding the composition of steps as general as possible to avoid coloring our perception and reduce the potential for bias in future decisions. The method helped us to identify the need of a work-in-progress state in the system, allowing the users to store "unfinished" fields. Time spent on the process of creating and refining a field can vary depending on the complexity of the task the field is supposed to perform. This state would allow users to exit the interface in the progress of creating a new field, and return to the same field at a later point continuing the process from where they left. Additionally we identified the need for evaluation and validation functionality, as a means for users to acquire feedback about the field and prevent faulty to be saved into the application. The resulting flowchart was used as a reference during the design of first iteration of scenarios and construction of prototypes.

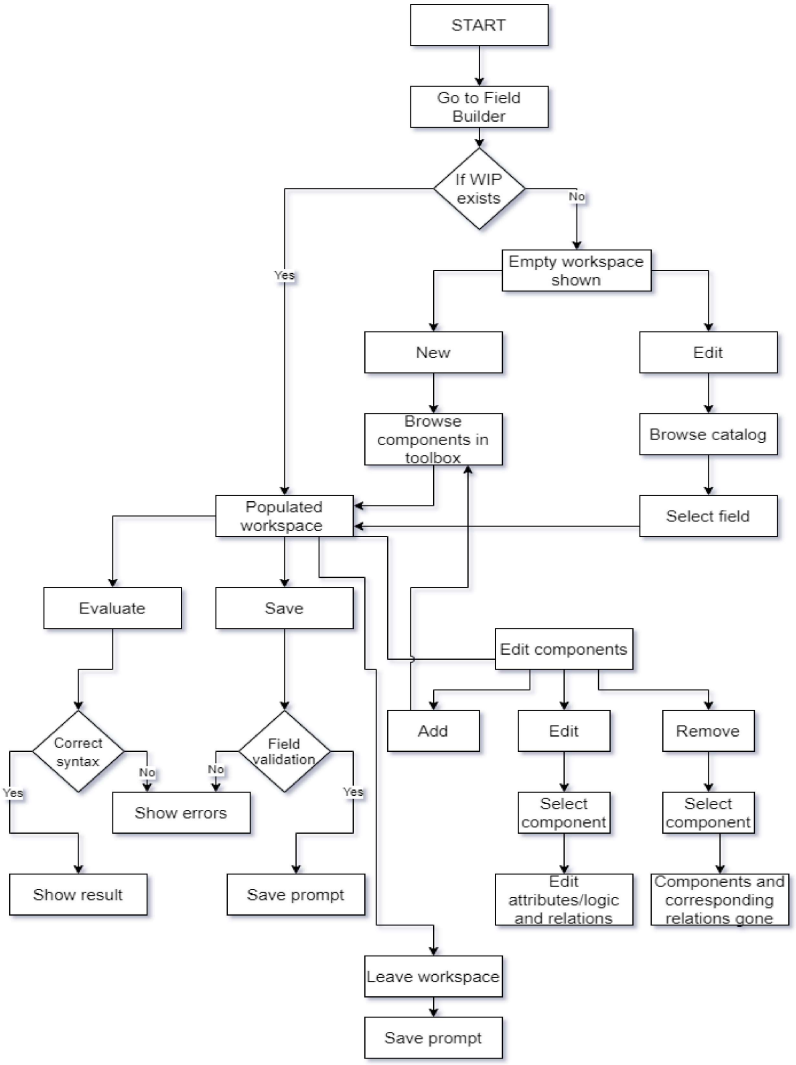



**Figure 3.3:** The flowchart constructed during the first phase

This method served an important purpose in this project as it allowed for visualization of possible user actions and their consequences. Additionally it is fitting for processes that have a distinct start and end point, as is the case with this project. For creation of the flowchart a valid starting point would be a user with intention on extracting particular data from the database, that is not available elsewhere in the application. Consequently the user will attempt to describe the constraints of relevant data through the graphical user interface. An end point for this scenario would be the presentation of requested data to the user.

### 3.2.3 Data-driven User Analytics

In the literature we studied, this method was referred to as "Site-search Analytics", describing its application primarily within the context of analysing data logs of search queries that users submitted. We decided to expand the application of this method to include a broader scope of analyzed data, as we discovered that there was additional, useful user data available other than used search queries. Notably information about users profession titles was available, allowing us to better understand the distribution of the target user base.

The company uses an data visualization tool called Kibana [30] to store usage data, that acts on top of the ElasticSearch [31] engine. ElasticSearch is a NoSQL database optimized for full-text search and data analytics, that allows for real time analytics and searches in big volumes of data.

One particularly insightful source of information were customer data sets which contained user-made tags. As tags are a precursor in functionality when compared to a field, this data was highly relevant. Analysis of these tags revealed the most common use cases, with a major emphasis being on retrieval of static values. This knowledge served as a baseline for design of scenarios in the later phase of the design process. Our initial goal is to make sure that the common baseline use cases are covered before introducing new features and capabilities to the system.

The second most used tag was tags that fetches values from related resources. These tags are primarily used to include tags from another type of resource into the the another resource context of the table explorer. For example if a user wants to show a file system related tag in the context of the resource type "machines", a tag that acts as a bridge between these two resource types must be made. A typical use case could be that a user wants to know how much storage volume that is available in a file system hosted at a particular server in their IT environment.

One interesting aspect was that some customers used filters very actively, while some customers barely used them at all. Filters are additional constraints users can enforce on tags. Our conclusion regarding this was that the UI for this feature is convoluted by being hard to spot and to understand. Users that have participated in an introductory workshop for the program and have been introduced to this feature and therefore know that it exist while other users don't know about it's existence. However, we could not find any data backing up this hypothesis. Although, due to its active usage among some users, we can conclude that it's an important feature that we want to make more available in the new interface.

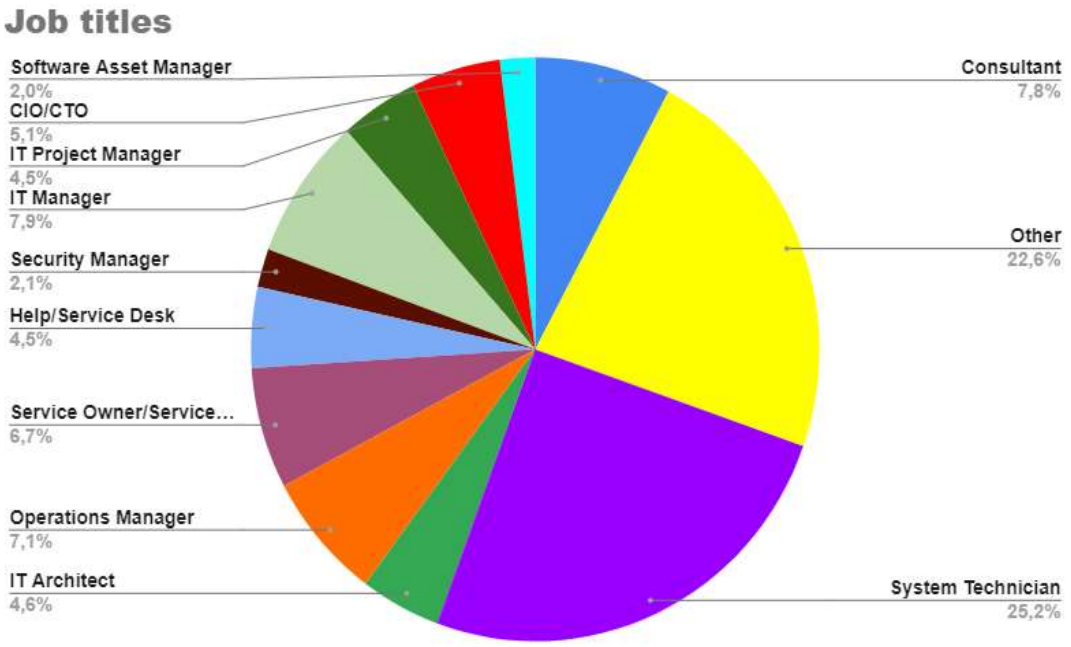



**Figure 3.4:** Distribution of job titles among users

## 3.3 Phase two

In the second phase of the design process, effort was put in into gaining an understanding of target users by performing card sorting, conducting interviews, competitive analysis, task analysis, creating personas and defining scenarios.

### 3.3.1 Card Sorting

The goal for this sorting activity was to understand how the users perceived the workflow through grouping different features. Having a clear objective helped us to properly adapt the characteristics of method.

To get some initial card ideas we partially extracted the main features from the previously created flow map that described the most essential interaction sequences, as well as imported concepts from the concept map. The first set of cards consisted of terms that described both features and content. Feature card are cards the describe user actions and workflow, while content cards describe building blocks that user interacts with. Combining two different categories of cards is highly discouraged according to Spencer [32], since it can be confusing for the participants. Therefore we divided the set of cards into two separate decks with one consisting of feature cards and the other of content cards. When we reviewed the decks, we found the feature deck to be of greater relevance for the scope of this thesis. Since the prototype would not contain most of the available content, the content deck was discarded for this design method and left for future work instead when it would be more relevant. See table 3.1 for a list of feature cards that were created and used for this method.

From the deck consisting out of features we explored how users related to different features in terms of similarity, functionality and their thoughts about the interaction flows. To reduce any potential bias from our preconceived interaction sequences we reformulated card titles to be more general. After the first walkthrough of the deck we noticed that words like *create*, *edit* and *browse* were used very frequently. To address this we renamed the cards a second time using synonyms of those words to avoid grouping bias based on semantic similarity. Additionally we wrote a short description for every card to provide clarity to the features they represent and to act as a reminder during the sorting activity. We chose to keep the word *browse* since it is a commonly used and understood term for describing the process of searching through a set of available elements. See table 3.1 for the final list of feature cards. Description was omitted for cards that were sufficiently self-explanatory

After some consideration we opted for conducting the card sorting on an individual basis, instead of using groups of two test subjects in each. Reasoning behind this decision was based on multiple aspects. First, as Spencer states[32], in a group setting a risk exists for a dominant group member to control the process, limiting the other member's impact on the process. Secondly, with a limited access to test subjects, this approach offered an increase in data points leading to a wider range of output from this method.

The group of test subjects consisted out of developers, a content manager and the product owner. In total eight test subjects participated in this process. Each session lasted about 30 minutes for the main sorting phase and 15 to 30 minutes for the followup discussion. Since this method operated on domain specific knowledge which only the selected groups possessed, we did not include customers as it would require a lot of effort in explaining the terminology and the way of thinking when designing traversal algorithm with steps and other components. Further, such an explanation could introduce a bias in regards to the approach when users performed the task.

We applied a variation to this method called *open card sorting* [32], meaning that users were offered the option to write and add own cards or remove any of the pre-existing cards. This was done to enable more freedom in the process and to create the possibility for users to express their own thoughts and ideas. There was some variance in the degree to which users utilized this option. Users could thus be divided into two categories. In one category users added between 0 to 3 cards. This group consisted of back-end and front-end developers, who generally have a more functionally oriented approach. The other category included users that added between 5 to 10 cards. Users in this group had lower programming knowledge, and thus focused more on workflow aspects of the interaction. They created cards that described particular stages in the procedure, as well as cards that defined their view of what should happen next.

Before the start of each card sorting session we gave a quick rundown of each card and its meaning. We decided to add this aspect as a supplement to the short descriptions below each card title. This was done to avoid any confusion or misunderstanding about a cards meaning during the sorting, to allow the user to focus on the sorting without having to interrupt their thought processes to ask for clarification about a card. The challenge here was to present the cards in a way that did not create any bias in how the user interpreted and categorized the card based on our description. To avoid this, we presented the cards in a order where closely related features from our flow chart were not presented in a consecutive sequence.

**Table 3.1:** List of feature cards created by us

| List of feature cards | |
|---|---|
| **Browse existing field configs** | Get an overview of all existing fields in product |
| **Browse existing relationships** | Get an overview of all existing relationships in product |
| **Browse steps to use as a traversal building block** | Get an overview of all available steps to use in a traversal |
| **Browse relationships to use as a traversal building block** | Get an overview of all available predefined relations to use in a traversal |
| **Open an existing field config for editing** | Get an overview of all available fields to use as a component in the field you're building |
| **Open context of an existing relation for editing** | Visualize the components and attributes of an existing field config |
| **Build new traversal** | Visualize the components and attributes of an existing relationship config |
| **Construct new static traversal** | Visualize the components and attributes of an existing field config |
| **Define new relationship config** | Combine steps and relations to create a static traversal |
| **Create new field config** | Combine steps and relations to create a relationship |
| **Modify field config** | Define the `id`, `name`, `description`, `cardinality`, `fieldtype` ... for a new or an existing field config |
| **Select rule ordering in the context of a field config** | Define the ordering and the `stopRule` attribute for the individual traversals for a new or already existing field config |
| **Populate step** | Select attributes for the specific step.For example `label` for `VertexHasLabel` and `fieldId` for `VertexFieldStep` |
| **Modify relationship config** | Configure the out and in constraints and traversal functions for a relationship config |
| **Construct Traversal** | Define the interrelations between building blocks e.g steps, relations |
| **Save field config** | |
| **Save relationship config** | |
| **Evaluate current field config** | Check for inconsistencies and errors in the field config active in the working area |
| **Evaluate current relationship** | Check for inconsistencies and errors in the relationship config active in the working area |
| **Delete existing field config** | |
| **Delete existing relationship config** | |
| **Reset workspace** | Clear your working area |

At the beginning of the session we inquired about consent regarding an audio recording of the session and clarified that it would be only be accessible and used by us and that it would be removed upon completion of the thesis work. At the end of each session the final state of the layout was photographed for use as a reference at a later point.

During the initial introduction the users were instructed to vocalize their process, to facilitate the *think aloud protocol*. On some occasions users were reminded to explain their thought process, especially when moving cards between groups. This is an integral part to the whole method and proved to be very insightful regarding understanding of the user's mental model of the system.

There was some variety to the method users took in regards to the mental process they followed when sorting the cards. Some users applied a functionality driven approach. These users created groups primarily based on the functionality represented in the semantics of the card - for example all cards mentioning traversals were grouped together into one pile and all cards mentioning field configurations into another. Initially this resulted in few, but large groups. After the user was satisfied with the layout, we had a short discussion where they had to motivate their groupings. Afterwards we inquired them to split the groups further, into smaller stacks, in an attempt to dive deeper into their understanding of the cards presented. The resulting groups in this sorting approach gave us insights regarding how we could group the different actions in the interface e.g. where actions like buttons and menus should be placed in relation to each other.

Other users applied a work-flow oriented approach, structuring the groups in a sequential layout, placing cards with regards to the phasing of when they would be relevant. For example cards with "create", "define", "open" were grouped and on the left side of the area, as if along an implicit time axis. Similarly, cards with "construct", "modify", "delete" were placed in a group following the first. We found this approach of sorting to be very insightful and gave us an good understanding of how we should design the action sequences in the interface, in contrast to the action grouping in the functionality driven sorting approach. The action sequences derived from this design method allowed us to perform a less comprehensive task analysis later on in the design process.

When the subject had reached a final state of the grouping and felt done with the sorting we continued with a open-ended discussion where we discussed more in-depth aspects of the features and how a potential implementation could be designed. The outcomes from the discussions provided us with very valuable feedback, especially from front-end developers, who were experienced in the field of interaction design and UX. These discussions gave us a great foundation when we began the creation of the Lo-Fi prototype.

### 3.3.2 Interviews

A goal with the interview was to extract the user's mental model of the product. A mental model is a user's perception of how something works [33]. It is vital for us as designers and researchers to understand how a user perceives the functionality of the product from a conceptual point of view and the importance it plays in their user experience. Further insight about existing patterns of behaviour can be gained and used in following stages of the design process, increasing chances of designing an intuitive product.

In total five interviews were conducted remotely with the aid of video streaming software Microsoft Teams [34]. All sessions lasted in the range of 45 - 60 minutes and were

**Table 3.2:** Set of interview questions

| Interview Questions | |
|---|---|
| **#** | **Question** |
| 1 | What is your background? |
| 2 | What is your role within the organisation? |
| 3 | What value does the product provide for you personally? |
| 4 | How frequently do you create new tags? |
| 5 | What is a typical goal task in mind in creating new tags? |
| 5a | Who is responsible for creating new tags? (Follow up question in case the subject has no experience with tag building) |
| 6 | Do you have any tags in mind that you have not been able to create using the current interface? |
| 7 | From your perspective what are the capabilities of the tag manager? |
| 8 | What is your understanding of how the product operates behind the scenes? |
| 9 | From your perspective what are the capabilities of the tag manager? |
| 10 | Thank you for your time. Do you have any questions or is there anything else you would like to discuss? |

recorded with the users consent. The participants were all male and between the ages of about 30 to 50. During some of the interviews some users offered to share their screens to facilitate the ongoing discussion. This aspect was not something that we had initially planned on including in an interview, but we welcomed that offer, as it created an opportunity for us to observe how a user performed tasks and navigated in the product, which was useful input when we performed task analysis in a later stage. Since we are expanding and improving upon an existing functionality, the user's mental model will serve as a framework for building the conceptual model of the new version.

A list of questions is presented in table 3.2. The first three questions were intended to provide brief background information about the interviewee, their technical skill level, experience with the product and role within their organization. This is critical for understanding different types of users and their needs. Data from the interviews was used in creation of personas, to give them aspects grounded in reality. Questions 4-9 serve to give an understanding of how the users engage with the current version of the product from a practical perspective.

Once the screen sharing session started the focus shifted to the user's current impressions and experience with the table explorer and the creation process of tags. From this point on the interview format shifted to an unstructured one. We asked the users to show some of the tags they had created and inquired about their experience with the interface for performing the tasks.

One important point brought up during one of the interviews is that the user was aware that the sought after information is stored somewhere in the system, but they were unsure how to actually access it. The perception of availability of information is partly gained from the introductory workshops held during the trial period. Users have the impression that the product in question is highly sophisticated and trust that it manages to properly collect large amounts of data about user's company IT-infrastructure.

Occasionally users made general comments regarding the interface and present func-

tionality. Some of these comments were made about features that were outside the scope of our work, but the feedback was valuable nonetheless, as it gave us a deeper understanding of the user's thought processes and perception of the product. For example, on multiple occasions general issues with the interface were brought up, such as difficulty finding a certain feature. Some users stated annoyances along with suggestions about potential fixes. These comments were relevant from a usability perspective, as they pointed out various design flaws. Such comments were noted and forwarded to the development team for further investigation.

However one comment had a particular relevance for our work. A user mentioned the absence of a progress indicator for the current state of the system in form of a loading icon. This was encountered in the context of working with larger sets of data in the table explorer, which could take time in a range between 5 to 15 seconds. From his previous experience he knew that the system was busy, but he expressed a desire for a loading icon or a progress bar. This could be relevant for iterations in future work when the prototype will be integrated with the back-end, which could result in waiting system states.

What became apparent after conducting the interviews is that users have a very narrow mental model, that in most cases was confined to some conceptualization of a typical database system.

### 3.3.3 Competitive Analysis

The main goal with this method was to identify aspects of similar products where they fall short, and aspects that makes them shine. Although this method can be applied in any stage of the design process, we chose to conduct it in the second phase, as surveying tool to establish additional guidelines against which to measure our own progress [10]. While this method can be applied in a wide scope, covering financial, demographic and marketing aspects, due to the nature and goal of this thesis we narrowed it to focus on the end-user experience. Kuniavsky suggests using focus groups in application of this method [10] and while such use can provide greater insight, due to current pandemic status we opted for conducting this method ourselves [35].

Our first idea was to study query building tools specifically for graph based databases but we failed to find any candidates that fit that specific criteria. We theorised this fact being due to the domain of graph databases being relatively new, and due to no established tools having yet reached the market. Instead we opted for studying other visual programming tools available. While the products selected for this method are not direct competitors to the host company of this thesis, they were still relevant for investigation in the context of interaction design.

To better compare the different solutions to each other we defined a simple scenario, where the goal was to output a message containing the number of times a certain button has been pressed. This algorithm was implemented on every candidate that supports this type of user action.

#### Benchmark framework

We constructed a simple benchmark framework for evaluating different products. We chose the following key aspects to keep in mind when analysing the products. Products

with particularly good solutions to these aspects will be noted and potentially incorporated into our prototypes in the next phase. We will also look after commonalities across the products to identify common design practices in the field of visual programming.

- **Color coding** - functionality that allows the user to manipulate the graphical appearance of elements in the workspace. For example changing the color of a node, port or a connection between two ports.
- **Comments** - the option to add comments, i.e. non-functional elements with the purpose of describing, clarifying or documenting a certain part of the graphical schema.
- **Content Aware Filtering** - ability of the interface to change the available elements depending on their compatibility with elements currently in use.
- **Input validation** - implies forced constraints on user actions. Upon any change in the graphical schema, the change is checked for consistency to prevent user from creating infinite loops, other fatal errors and preventing connection between incompatible types of input and output ports.
- **Instant feedback** - upon a user initiated action, the interface automatically updates and re-evaluates the graphical schema and shows the result.
- **Organizational tools** - the ability to visually organize different parts of the workspace.
- **Required Programming Knowledge** - the recommended level of understanding of different programming concepts and paradigms to be able to use the tool efficiently. This requirement increases with functional richness and complexity of a tool, and can be decreased to some degree by applying abstraction layers to the interface.
- **Functions**
  - **Resize** - allows the user to modify the workspace layout.
  - **Search** - allows users to search for a specific element type among the available ones by defining a query in a search bar.
  - **Zoom** - allows the user to increase or decrease the zoom level of the workspace.

## Blockly

Blocky is a programming library for creating JavaScript code using visual programming [36]. It is open-source and maintained by Google. Blockly is often described as a block-based VPL, meaning that the syntax and the available operations are visualized by blocks. The blocks have differently shaped interlockings like a jigsaw puzzle. This design contributes to an increased affordance to intuitively show the user what combinations of operations is possible. For example, a regular if-statement block requires an expression that evaluates to a boolean variable, this relation is shown by a piece of jigsaw puzzle interlocking.

**Noteworthy design traits**

- **Comments** Each individual comment can be linked to a block. When a comment is added an icon is embedded into the block, which allows for showing/hiding the comment.

- **Input validation** Blockly achieves a good balance between enabling users to create algorithms that results in troublesome states and preventing users from creating algorithms that results in errors. For example, nothing hinders the user from creating infinite loops, which could result in the browser crashing. But on the other hand, loops that always are active could be a wanted feature when creating multi-threaded real-time applications or some forms of event listeners. By making it impossible to connect blocks that are not compatible they prevent the user from creating code that results in errors. Additionally they have found a great way to deal with type safety in an intuitive way. By leveraging a metaphor like puzzle interlockings that most people are familiar with, they achieve good usability for both experienced programmers as well as novices. However, the interface is lacking in clear affordance regarding compatible puzzle interlockings as only one type of interlockings exists. For example the *repeat while* block is incompatible with a *text string* block, even though the puzzle pieces seems to fit.

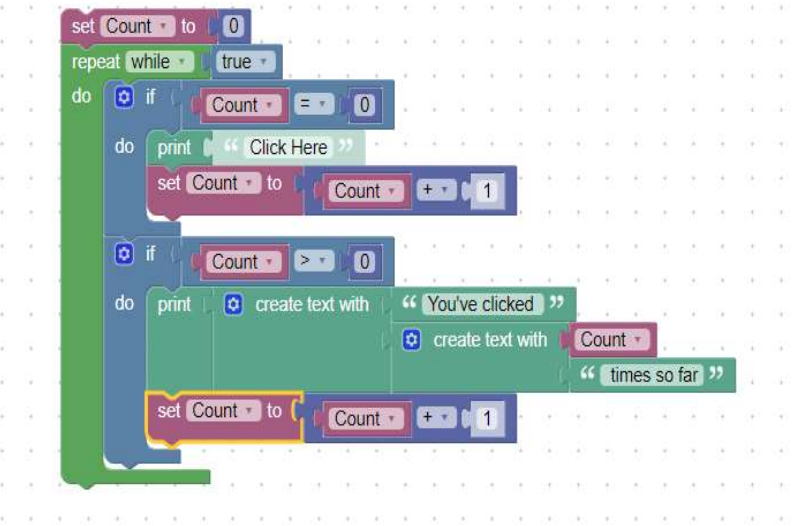



**Figure 3.5:** The algorithm implemented using Blockly

## Scratch

Scratch is a project started at MIT, with the intention to serve as a more inviting and "low floor" introduction to programming and computer science, targeted primarily towards children [18]. Scratch is fork of Blockly and therefore shares a lot of similarities. It is also blocked-based and employs constraints to show compatible expressions and statements. However, it focuses more on geometrical shapes instead of puzzle interlockings. The syntax vocabulary is embedded within blocks and has a higher degree of abstraction and con-

sists out of human-readable sentences instead of the typical if, do, while or for statements, that is typical for the most programming languages.

**Noteworthy design traits**

- **Required Programming Knowledge** Scratch really stands out when it comes to the amount of prior programming experience needed to get started. As mentioned in the description scratch achieves this partly by using human-readable sentences as syntax, and partly by the geometrical constraints applied on various blocks.

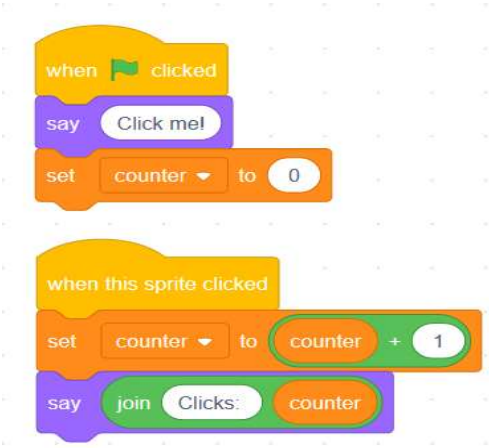



**Figure 3.6:** The algorithm implemented using Scratch

## Kodu

Kodu is a visual programming language created for developing games for the Microsoft platform, in particular for Xbox [19]. It's main target audience is children and thus requires no previous programming knowledge. The logic and syntax are completely icon-based. Programs consist of pages, where a set of rules are defined. All the rules are based around specific scenarios and consists of a condition and action part. The condition could for example be that a arrow button has been pressed and the action could be to move one character in that direction. All the conditions are evaluated simultaneously, making it simple to create larger programs.

The scenario created for evaluation of different products was not used in Kodu since it was not applicable due to a limited instruction set. Instead we explored the general workflow of the product.

**Noteworthy design traits**

- **Content Aware Filtering** The interface for defining rules adapts to the content, depending on which conditions and actions are currently applied on the current row. This prevents the user from creating meaningless rules or ones that could potentially lead to errors.


**Figure 3.7:** Kodu rule creation interface

## FlowHub

FlowHub is a visual IDE for flow-based programming with capabilities for full stack development [38]. Predefined components are available for a large set of common programming tasks. Components can also be created by implementing them using JavaScript. In contrast to the previously mentioned visual programming software, FlowHub is more suitable for professional and large scale development, where you have to organize large applications, making complex workflows and asynchronous processes. While the framework is very powerful and flexible, it's considerably more advanced and has a steep learning curve making it more targeted towards already experienced programmers.

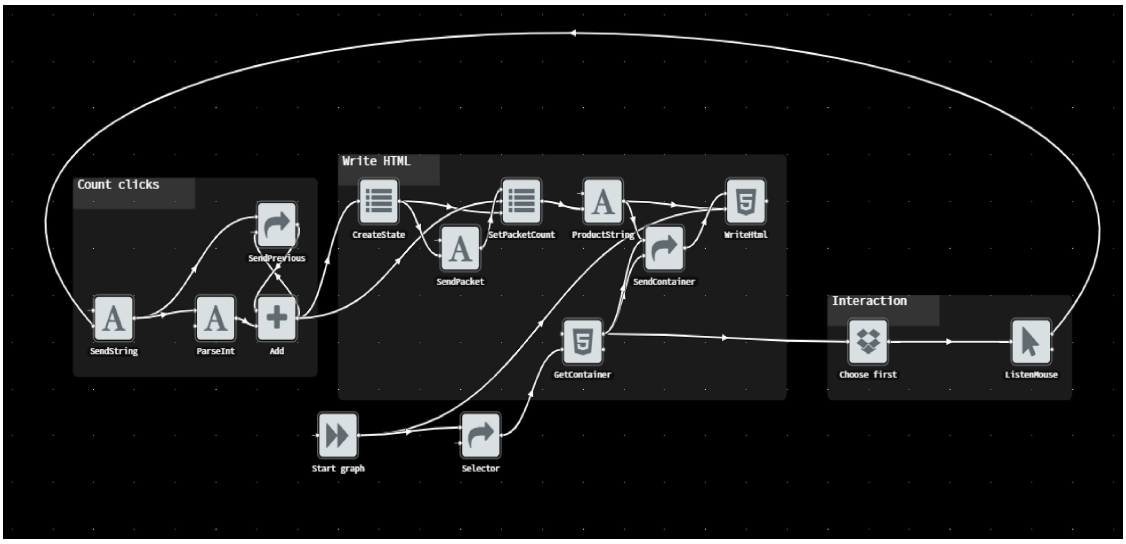



**Figure 3.8:** The algorithm implemented using FlowHub

### Noteworthy design traits

- **Organizational Tools** Just like many textual IDEs FlowHub offers tools for organizing and visually structuring the code. For example the ability to create packages

in the form of rectangular boxes covering a certain area containing two or more components. See figure 3.8 for an example.

## Flowgorithm

Flowgorithm is a graphical tool for designing and executing flowcharts [37]. It offers a high degree of abstraction and focuses on the logic behind the algorithm rather than the syntax, for that reason it reminds of pseudocode. Just like flowcharts, the statements and expressions are linked together by edges that defines flows or relations.

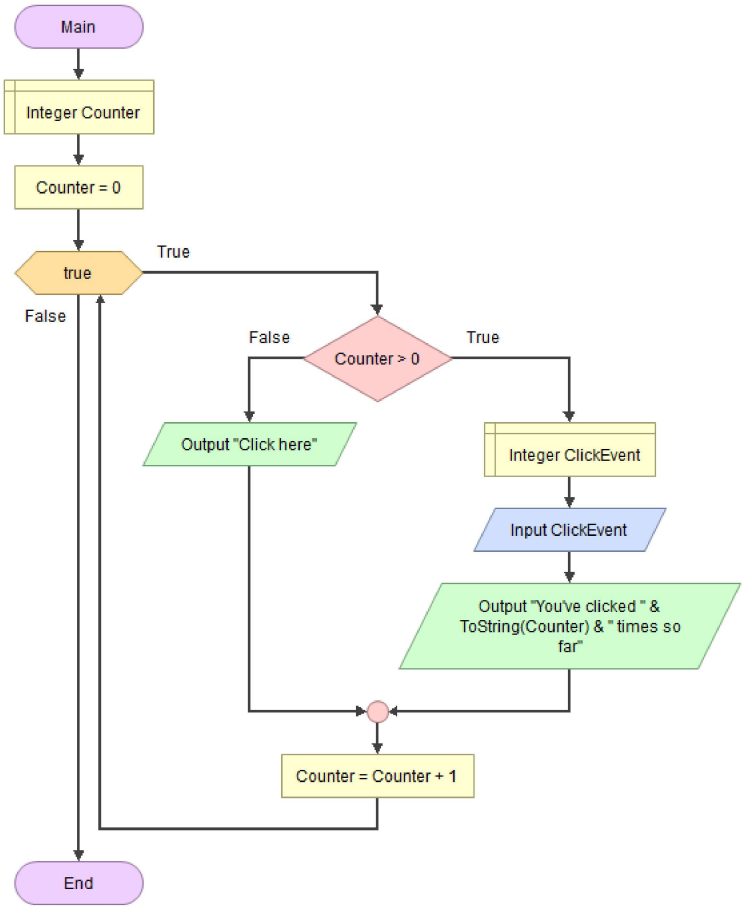



**Figure 3.9:** The algorithm implemented using Flowgorithm

### Noteworthy design traits

- **Overview** Flowgorithm provides a very clear overview of the execution path from the beginning of the program, signified by the *Main* node, to the end of the program, signified by the *End* node. See Figure 3.9 for an example.

### Common Design Traits

- **Color Coding -** Every product featured color coding in some form. For example Blockly, Scratch and Flowgorithm applied color coding to the nodes, to signify their type, while FlowHub allowed users to customize colors of the edges connecting nodes.
- **Comments -** Blockly and Scratch offer a way to add comments to the code by appending special comment blocks to the workspace.
- **Organizational tools -** All products, except for Kodu, offered the user options to manipulate the workspace, for example by dragging the visible work area along the vertical and horizontal axes or zooming in and out. This adds an additional dimension to formatting the code, compared to only the vertical in regular IDEs.
- **Input Validation -** All products have some form of input validation, with Kodu having a particularly good implementation by applying content aware filtering which forces constraints on available elements.
- **Instant Feedback -** While the program is running, any change made to the program is evaluated and applied in real time. FlowHub shows errors by highlighting a conflicting relation or element with a glowing red border.

## 3.3.4 Task analysis

The company routinely conducts workshop sessions where users get assistance with issues or implementation of particular use cases. A representative from the company with deep knowledge of the application's functionality facilitates the process. The context of these sessions ranged from educating newly acquired users to helping already established ones. The sessions are conducted remotely, with the use of screen sharing and voice communication software. As the scope of this project is improving and expanding upon the functionality of an existing product, the purpose of this method is aligned well with our goals.

We have chosen to perform this design method in a slightly less comprehensive way than the guidelines from the literature. The literature proposes a broad method that includes creation of a visual representation of tasks, sub-tasks and key decisions, for example in the form of a flowchart or some other graphical schema. We found that approach to be unnecessary to extract value from the gathered data, thus we opted for a more barebone approach, without applying the graphical part. We also found it to be more efficient to use this design method as a complement to other, already conducted methods, such as interviews and card sorting.

We had several opportunities to sit and observe these sessions to gain an understanding of user's problems and thought processes, as well as an insight into various use cases. We sat as silent observers, taking notes and after the session discussing and reflecting upon the gathered material.

Some observed use cases required the use of regular expressions, commonly referred to as regex, in the filter function. Regex stands for "regular expression" which is a common notation used to describe a pattern of characters used in string searching algorithms [39].

Primarily it is used to define constraints on strings, for example "Show only strings that contain three letters followed by two numbers". While the filter functionality supports regex, it does not indicate in any way that it does. This potentially leads to an increased amount of support requests from the users, as they are not aware the possibility to apply regex in their filtering. Users with some technical background were able to deduct from the filter functionality that regex was applicable. This functionality could benefit from clarification, as regex is a powerful feature that enables implementation of more complex fields. Further, it would potentially reduce the amount of support inquiries from customers.

After conducting interviews and task analysis it became apparent that users had a high degree of familiarity with their own data set. This was an important insight, as it supported the need for a functionality in the debug mode which would allow the user to select an actual asset in the database as an input to the field, to evaluate the field for validation of the result.

One of the main outputs from workshop sessions was data about common use cases. After a couple of sessions it was possible to identify some common usage patterns. These patterns served as a basis on which we constructed scenarios, which would be used for testing and evaluation during later stages of the design process.

In addition to observing the end users we also implemented a field programmatically to gain first hand experience with the general work flow and obstacles in the process. We received some assistance from developers, where they shared some advice on the approach we could use based on their own work flow.

Our insights from this activity were that the creation of fields often involves an exploratory phase were the developer navigates the graph to get an understanding of how the requested data relates to the resource type it will be applied on. The focus is on which edges and vertices that needs to be traversed in order define the path in the graph. This is performed in the console interface to the database using gremlin. Is some cases this step is unnecessary when the developer in question is aware of the specific part of the graph that the traversal will be applied on, but usually the names of vertices, properties and edges needs to be checked so they get defined correctly in the traversal. Another common aid when creating fields are by taking inspiration from functionally similar fields and how they are constructed. Currently many old tags are being converted into fields, in this case the gremlin code in the tags are the main reference.

### 3.3.5 Scenarios

To leave room for integration and exploration of insights in future design methods we have chosen to define the scenarios broadly in this first iteration of this method. The primary purpose of this approach is to contextualize a potential workflow for use as a reference when proceeding to the prototyping phase. At an early stage it is beneficial to avoid being too specific in description of scenarios and other user flows, to avoid bias based due to early ungrounded assumptions [40]. Therefore following scenarios are not representative of an actual workflow that will be present in a later iteration of the prototype.

To facilitate generation of scenarios we drew rough sketches of a potential interface rooted in conclusions from flowcharts and task analysis. Additionally we used insights from competitive analysis with regards to the basic workflow.

With reference to the pseudocode of a field presented in chapter 2.3.4, each property

that contains a nested element is available as a graphical component used in the workspace to construct a graphical schema.

We have defined four scenarios for distinct types of users in mind. Two basic scenarios targeting a user with close to zero knowledge of the inner workings of the application, that are extracted from the most common use cases today, found through the data-driven user analytics design method. The intermediate scenario is created for content managers at the company with some basic understanding of the application's capabilities and functionalities. Finally, the advanced scenario targets an in-house developer with experience of the how the graph database is structured and possesses knowledge of all of the available functions.

***Scenario 1 - Basic:*** This scenario focuses on how to create a field that imports a static value, synonymous with the fixed value tags that exists in the current application. See table 3.3 for details.

*Background & Task*
A typical goal for creating static value fields is importing data that is not accessible by the product inventory methods and mapping it to existing assets in the product. However, it is available at a source defined by the customer, for example an Excel spreadsheet. This is usually static information that has to be defined by a user, for example "Server X is located in room Y". The user want to assign this comment to the "Server X" asset.

*User goal*
To create a field that retrieves a static value representing a comment about a specific server.

**Table 3.3:** Basic scenario

| Scenario 1 - Basic | |
|---|---|
| **Step 1** | Niklas needs to create a new field that shows non-inventoried data |
| **Step 2** | He opens up the Field Builder interface |
| **Step 3** | He adds a field configuration component to the workspace and populates it with metadata e.g. name and description. |
| **Step 4** | He adds a constraint component to the workspace and specifies "Server X" as the constraint |
| **Step 5** | He connects the constraint component to the field configuration component |
| **Step 6** | He defines "computersystem" as the constraint component's property, to include servers in the scope of the field |
| **Step 7** | He adds a function component and connects it with the field configuration component |
| **Step 8** | He adds a static value component and connects it with the function component |
| **Step 9** | He sets the property of the static value component to "Server X is located in room Y" |
| **Step 10** | He saves the completed field and validates it's functionality in the main product |

***Scenario 2 - Basic:*** This scenario focuses on how to create a field that fetches a Value from related resource, synonymous with the *value from another resource* tag that exists in the current application.

*Background & Task*
Mats have created a new table that lists all virtual machines in the organisation's IT-environment. Every Virtual Machine (VM) in her organisation's infrastructure has a responsible service technician and for each of them she wants to list the service technician's contact information in the table. See table 3.4 for details.

*User goal*
To create a new field that links together two different resource types (User account and VMs) and then fetches the email addresses for the responsible service technician for every individual VM.

**Table 3.4:** Basic scenario

| Scenario 2 - Basic | |
|---|---|
| **Step 1** | Mats needs to create a new field to fetch the email addresses of responsible service technicians for every Virtual Machine |
| **Step 2** | He opens up the Field Builder interface |
| **Step 3** | He adds a field configuration component to the workspace and populates it with metadata e.g. name and description. |
| **Step 4** | He adds a constraint component and selects "virtualmachine" as its property and connects it to the field configuration component |
| **Step 5** | She add a function component and connects it with the field configuration component. |
| **Step 6** | He adds a relationship component and selects vm-useracount as its property and connects it to the function component. |
| **Step 7** | He adds a property component to fetch email addresses related to user accounts, she therefore selects emailaddress as the property for the component and connects it to the relationship component. |
| **Step 8** | He saves the completed field and validates it's functionality in the main product |

***Scenario 3 - Intermediate:*** This scenario focuses on how to create a field that filters all applications installed on a specific machine and looks for the application "Zoom", if found the field will return the boolean value true else false. See table 3.5 for details.

*Background & Task*
Niklas works as a content manager at the company. He receives a request from a customer for a specific field. They wants to retrieve a boolean value of every machines in the company's infrastructure that have the application Zoom installed. He has already browsed through all the existing fields and found no one suiting this use case. Therefore he decided to create a new field for his purpose using the Field Builder.

*User goal*
To create a new field that displays true or false whether Zoom is installed at the machine.

**Table 3.5:** Intermediate scenario

| Scenario 3 - Intermediate | |
|---|---|
| **Step 1** | Niklas needs to create a new field upon a request by a customer |
| **Step 2** | He opens up the Field Builder interface |
| **Step 3** | He adds a field configuration component to the workspace and populates it with metadata e.g. name and description. |
| **Step 4** | He adds a constraint component to the workspace and configure its property to be set to "computersystem" and connects it with the field configuration component |
| **Step 5** | He adds function component to the workspace and connects it with to the field configuration component |
| **Step 6** | He creates a filter component and configure it to allow values that contains Zoom and connects it to the function component |
| **Step 7** | He creates a property component and sets the property to fetch installed applications and connects it to the filter attribute in the filter component. |
| **Step 8** | He creates a static value component and sets it property to true and connects it to the filter component to be return if the filter attribute succeeds |
| **Step 9** | He creates a static value component and sets it property to false and connects it to the filter component to be return if the filter attribute fails |
| **Step 10** | He saves the completed field and validates it's functionality in the main product |

***Scenario 4 - Advanced:*** This scenario focuses on how to create a field that applies a filter that only allows values that are discovered the last 7 days. The field also utilizes two separate traversals to fetch the value from different data sources. See table 3.6 for details.

*Background & Task*
The content manager have asked Philip to implement a field based on a customer request.

*User goal*
To create a new field that displays the status of Virtual Machines inventoried form HyperV or VMware, that have been inactive for 7 or more days.

**Table 3.6:** Advanced scenario

| Scenario 4 - Advanced | |
|---|---|
| **Step 1** | Philip needs to create a new field upon a request from a content manager on the behalf of an customer |
| **Step 2** | He opens up the Field Builder interface |
| **Step 3** | He adds a field configuration component to the workspace and populates it with metadata e.g. name and description. |
| **Step 4** | He adds a constraint component and set its property to "virtualmachine" and connects it to the field configuration component |
| **Step 5** | He defines a rule configuration component and expands it to have two function components as input. Both gets set as stopRules. He then connects the component to the field configuration component. |
| **Step 6** | He defines a filter component that filters values based on when they were discovered, he configures the filter to only allow values that are discovered the last 7 days. He then connects the component to the rule configuration component. |
| **Step 7** | He creates a property component and sets it's property to HyperV status and connects it to the filter component as a filter attribute |
| **Step 8** | He creates a static value component and sets it to False and connects it to the filter component as the value that will be returned if the filter attribute succeeds. |
| **Step 9** | He creates a static value component and sets it to True and connects it to the filter component as the value that will be returned if the filter attribute fails. |
| **Step 10** | He repeats the processes between step 6 and 9 but selcets VMware status instead of HyperV status. |
| **Step 11** | He saves the completed field and validates it's functionality in the main product |

### 3.3.6 Personas

We applied this design method to establish a common ground for discussion about various user groups. This was done to concretize different user and their needs for us as designers, as well as for future reference for the company for the continuation of development of the Field Builder and other features. We have chosen to create personas for four different types of users from the targeted audience. These personas were derived from both an exploratory qualitative research with the interviews and task analysis as it's major data sources, and an statistical data-driven approach leveraging the findings from the data-driven user analytics design method. We also had an dialog with the sales department and the customer success about the their perception of the user base and found out that the a customer base analysis have been performed. The analysis pointed towards IT-technicians and service owners being the major professions among the users. This aligned well with the findings from the data-driven user analytics that showed that almost a quarter of the users were IT-technicians and another quarter were service owners or related professions such as IT, operations and software asset managers.

With this background we created the two personas representing IT-technicians and CIOs. The CIO were chosen to increase the diversity among the personas to include users that are involved with management and business aspects, but at the same time not discarding the traits of the service owners, since the two professions share similarities but still having distinct roles within an organization. Both roles have responsibility over the management of information and computer technologies with the main difference is that CIOs have a more high-level management and decision-making role within an organization. From our site-search analytic design method, we also noticed that this group made out five percent of the user base.

The scenarios and challenges for those personas were partly extracted from the interviews, where we explicitly asked them what value the product provided for them and had them show some common uses cases for the product. Additionally, by including observations from the workshops we attended in the task analysis method. The goals and needs of the personas were primarily grounded in the company's sales department perception of the customer.

The two additional personas, back-end developer and content manager represents the in-house users at the company that will use the Field Builder. Both these roles make up a small portion of the user base, but they will be the group of users who will use the tool the most, since creation of new fields is a common work task. These personas were grounded on our experiences at the company, as one of the authors behind this paper has been a part of and worked at the company in question for over a year as an developer prior to this thesis work. Therefore, we had a great understanding of the problem space and challenges with the current way of creating fields as well as the characteristics and skill set of the personas.

# MATS **BENGTSSON**

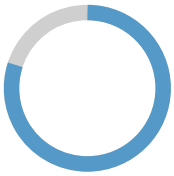
mats@bsns.com
Stockholm

Mats Bengtsson

## Background

Graduated with a bachelors degree in economy from Gothenburg School of Economics. Worked at several different companies after graduation and gained experience in the field of management. Has been working at his current company for 10 years. Started out as a business analyst and has been steadily climbing the corporate ladder to his current position.

## Skills



Business Management

Product Understanding

Technical Understanding

Programming Knowledge

## Goals

- Overview of organizational IT-assets for decision making regarding buisness aspects
- Increase of productivity
- Cost optimization

## Needs

Efficient process for information gathering, with little technical skill required.

## Challenges

Lacking technical understanding of the infrastructure.
Large organisational overhead with time consuming communication structure for gathering information regarding IT-assets.

## Scenario

Overview of license costs within the organization. The company has amassed large expenses related to license fees. Mats needs to investigate if the costs can be reduced by unsubscribing from redundant and unused licenses. The application provides insight by gathering information about all active licenses in relation to users and user groups.

# PHILIP NORDAHL
# BACKEND DEVELOPER

phlp@cpny.com
Malmö

Philip Nordahl
phildev

## Background

Graduated with a masters degree in computer science. Alongside his studies he accumulated programming experiences through summer internships and part-time employment at various tech companies. After graduation he started working at a junior developer position.

## Skills


## Goals

- Develop new functionality for the applicationt
- Streamline the process of creating fields so he can spend his time on more fulfilling tasks

## Needs

Get an overview of the large number of available options and the specific structure of the database.

## Challenges

The procedure of creating new fields is time consuming and cumbersome if one is not working within the context regularly, due to the specific programming syntax involved and rich naming convention.

## Scenario

The product manager requests new fields for a new integration of a data source. Philip needs to get it done within a few days before the next release of the product while he has other, more important tasks requiring attention.

# NILS HOLST
## IT-Technician

nisse@kommun.se
Kristianstad

Nils Holst
nisseshobby

### Background

Nils developed a passion for computers and technology early on in his life. He started out his proffessional career as an IT-support right after graduating from high-school, with a specialization in IT-networks. Eventually he started progamming in his free time, to open up the posibility of further advancement into a more technical position. After working as an IT-support for five years an opening for an IT-technician became available and thanks to his newly acquired skill he got the job.

### Skills


### Goals

Monitor the status of the IT-infrastructure in real time to facilitate proactive maintenance.

### Needs

Create fields that suit his specific use-cases, that are not covered by the pre-bundled packages of fields.

### Challenges

His options for creation of tags are limited by the currently available functionality. To create more advanced tags he needs to contact customer support and request it.

### Scenario

Nils is currently building a new report that provides an overview of the organizations servers. They are using Microsoft System Center Configuration Manager as an system management tool and have created individual description on every server concerning their physical location. Nils now want to include this information in his report and therefore he need to create a new field for displaying that value.

# NIKLAS LINDBERG
# CONTENT MANAGER

nkls@cpny.com
Malmö

Niklas Lindberg

## Background

Niklas graduated with a bachelor degree in science and information systems. He has a deep interest for the role technology plays from an organizational viewpoint. He also has a passion for human interaction and thus found the position of content management fitting his personality, as contact with customers is a vital part of the position.

## Skills

Business Management

Product Understanding

Technical Understanding

Programming Knowledge

## Goals

- Overview of organizational IT-assets for decision making regarding buisness aspects
- Increase of productivity
- Cost optimization

## Needs

Efficient process for information gathering, with little technical skill required.

## Challenges

Lacking technical understanding of the infrastructure.
Large organisational overhead with time consuming communication structure for gathering information regarding IT-assets.

## Scenario

Overview of license costs within the organization. The application provides insight by gathering information about all active licenses in relation to users and user groups. The company has amassed large expenses related to license fees. Niklas needs to investigate if the costs can be reduced by unsubscribing from redundant and unused licenses.

## 3.4 Phase three

In the third phase of the design process, our goal was to consolidate our acquired data and insight from previous phases and apply them in the implementation of a Lo-Fi prototype. We decided to create a T-prototype with our primary focus on the interaction regarding the construction of fields, which we assessed to be the most critical and essential interaction sequence. Aspects such as visual design, layout of secondary components such as toolbox, tabs and debug were deemed to be outside of the primary scope of this thesis. The construction process and user interaction related to it are areas that have the highest degree of complexity compared to other features, thus they warrant this choice of focus for the prototype.

This functionality was implemented with several interaction steps and made up the depth of the prototype. Other features were implemented with a single interaction step or just visible in the prototype but lacked any interaction. The features not covered in detail by the prototype will be dedicated an own section called *Reflections & remarks about features*, where we will present our thoughts and motivations behind the design of these features.

During this design phase new scenarios were defined with more detail to match the implementation of the prototype, and act as test instructions. The prototype was evaluated with user testing and iterated over until a satisfactory design and state was achieved.

### 3.4.1 Lo-Fi prototyping

Initially we began the Lo-Fi prototyping process in a more traditional way, by drawing UI elements on paper and cutting them out. A A3 white sheet was used as a backdrop on which the elements were placed. We used colored yarn strings to play the role of connecting links between the nodes. Input and output sockets were placed on nodes by using poster putty to allow for quick modifications, as well as easy reuse between testing sessions. Once we were satisfied with the initial iteration, we printed and cut out all the elements. See figure 3.10 for a picture of the paper prototype.

Shortly after we began the prototyping, we decided to use a design tool, Adobe XD, to create elements instead. The tool was used by a designer at the company to create mock-ups for the new interface of the upcoming product version. Using the tool he had created design guidelines that contained the color scheme, as well as all graphical elements such us buttons, input fields, icons, panels, etc. In this way, it was easier to visualize how the Field Builder would look and feel inside the product. Thus, the prototype fidelity is higher than the pen and paper approach. An illustration of the final Lo-Fi prototype created in Adobe XD is shown in figure 3.11.

An in-depth reflection about potential features is presented in the section following the evaluation of the Lo-Fi prototype.

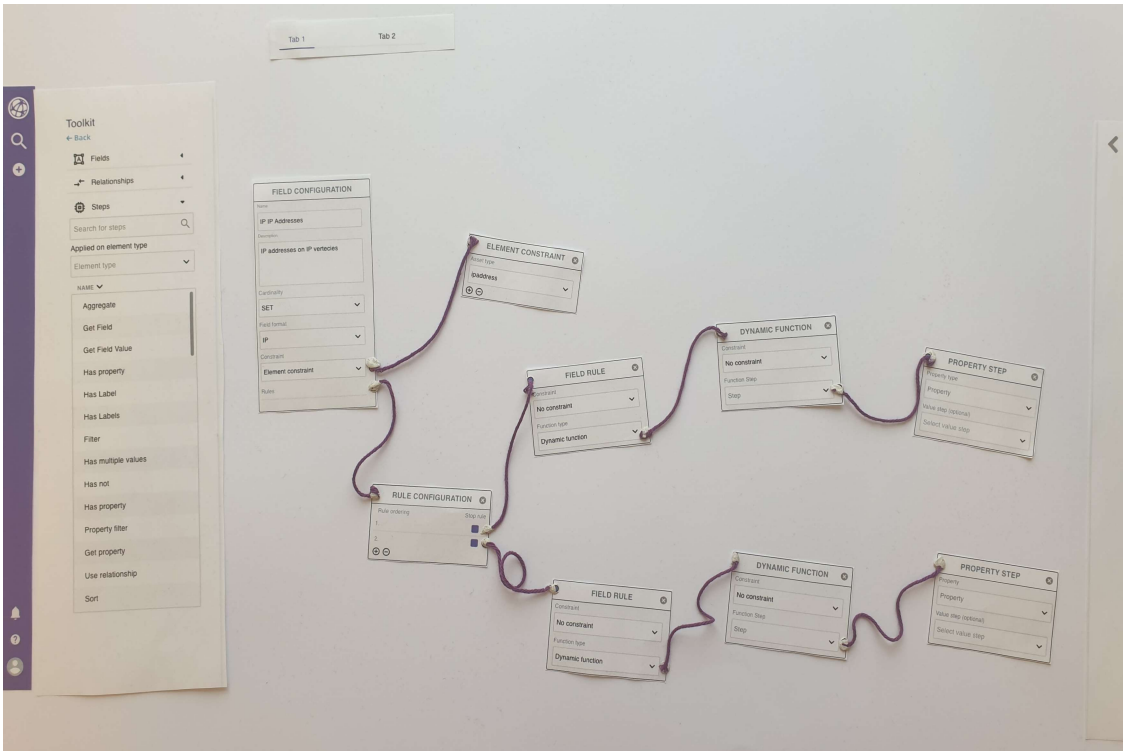



**Figure 3.10:** Lo-Fi paper prototype

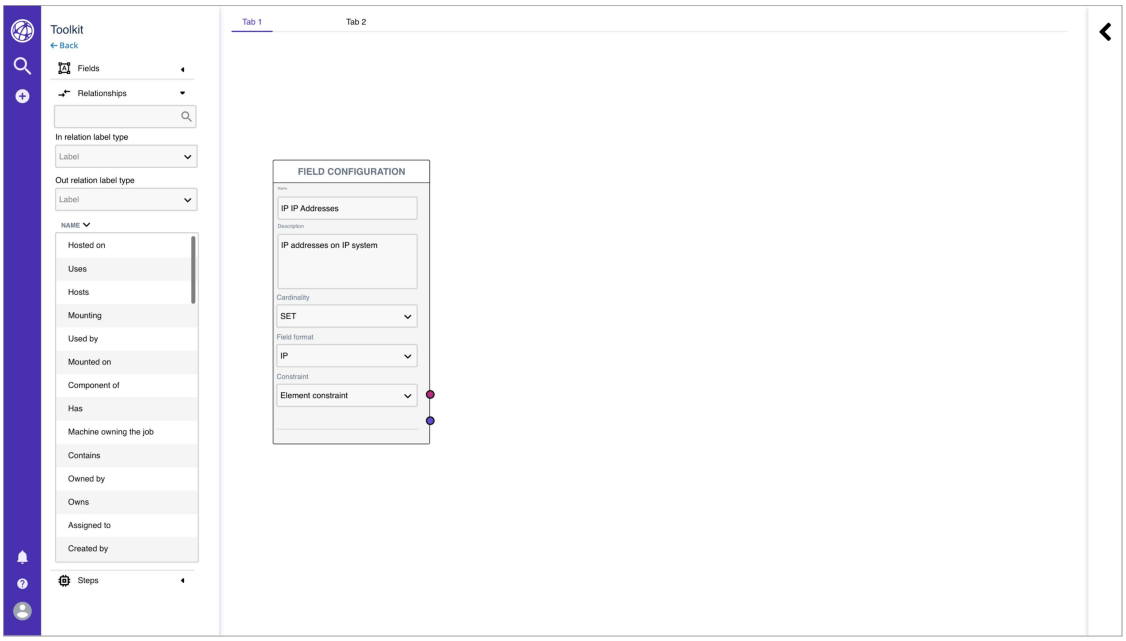



**Figure 3.11:** Lo-Fi prototype, created in Adobe XD

### 3.4.2 Detailed prototype scenarios

With artifacts and insights from previously conducted design methods we defined new and more detailed scenarios, that reflect the functionality and terminology used in the prototypes.

#### Creation of an IP-address field

*Background & Task*
This scenario focuses on the creation of a field that fetches both the IP-v4 and IP-v6 addresses in the database and presents them in a field. See table 3.7 for details.

**Table 3.7:** Creation of a field that counts IP-addresses assigned to a machine

| **Scenario 1** | |
|---|---|
| **Step 1** | Navigate to the Field Builder interface |
| **Step 2** | Populate the already existing field configuration in the workspace |
| **Step 2.1** | Choose a suitable name for the field, preferably "IP addresses" and type it in the name inputfield in the field configuration |
| **Step 2.2** | Add a description, for example "This field displays both ip v4 and v6 addresses" |
| **Step 2.3** | Set the cardinality to SET. (Because you only want unique values) |
| **Step 2.4** | Choose IP as the field format type |
| **Step 2.5** | Select "Element constraint" in the constraint dropdown menu to generate a new Element constraint component that are connected to the field configuration |
| **Step 3** | Select "ipaddress" as the asset type in the generated element constraint component |
| **Step 4** | Navigate and select the Rule configuration component in the toolkit and drag it into your workspace. Connect it to the rules socket in the field configuration component |
| **Step 5** | Navigate and select the field rule component in the toolkit and drag it to the workspace, then connect it to the rule configuration component |
| **Step 6.1** | Create an additional rule socket in the rule configuration component by clicking the "plus" button |
| **Step 6.2** | Repeat the previous step and connect it to second socket in the field rule component |
| **Step 7** | Set the constraint to "No constraint" in both of the field rules |
| **Step 7.1** | Select "Dynamic function" in the function type dropdown menu at the field rules to generate new dynamic function components |
| **Step 8** | Set the constraint to "No constraint" in both of the dynamic function components |
| **Step 8.1** | Select "Property" in the function step dropdown menu in both of the dynamic function components to generate new property step components |
| **Step 9** | Select "IPv4adress" as the property in one of the property step components |
| **Step 10** | Select "IPv6adress" as the property in the other property step component |
| **Step 11** | (Optional) validate the field in the debugger by choosing a asset type of "ipadress" as test object and inspect the returned values |
| **Step 12** | Save the field by clicking on the save action button |

The finished field should look like the field illustrated in figure 3.12.

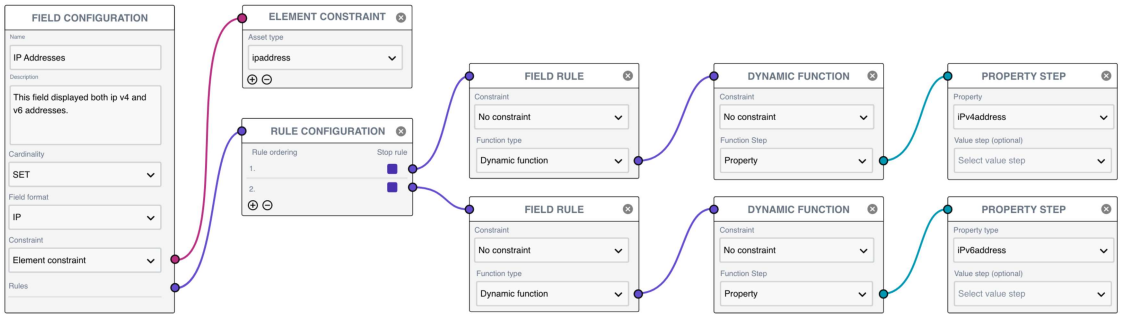



**Figure 3.12:** Implementation of the field in the scenario "Creation of a IP-address field"

## Creation of a field that counts IP-addresses assigned to a machine

*Background & Task*

This scenario focuses on the creation of a field that counts the number of assigned IP-addresses a specific machine have by leveraging the field created in previous scenario. See table 3.8 for details.

The finished field should look like the field illustrated in figure 3.13.

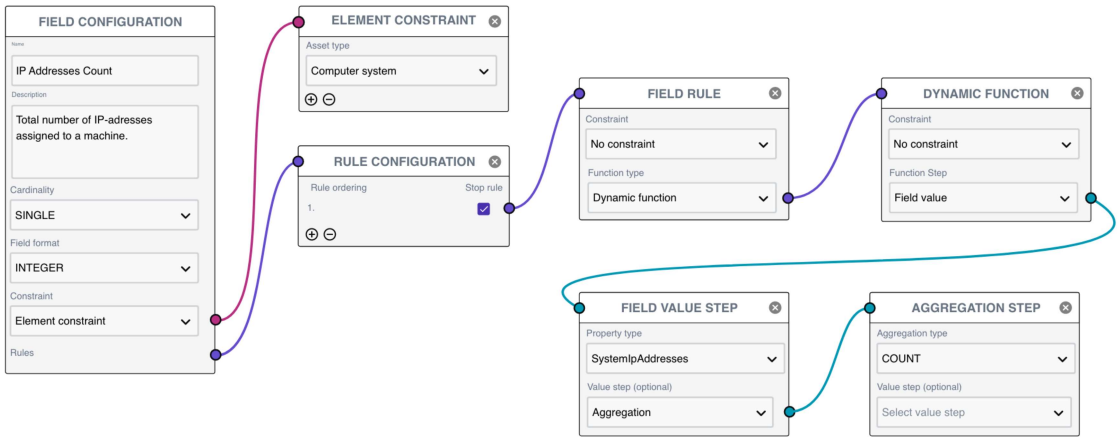



**Figure 3.13:** Implementation of the field in the scenario "Creation of a field that counts IP-addresses assigned to a machine"

**Table 3.8:** Creation of a field that counts IP-addresses assigned to a machine

| **Scenario 2** | |
|---|---|
| **Step 1** | Navigate to the Field Builder interface |
| **Step 2** | Populate the already existing field configuration in the workspace |
| **Step 2.1** | Choose a suitable name for the field, preferably "IP addresses count" and type it in the name inputfield in the field configuration |
| **Step 2.2** | Add an description, for example "Total number of IP-addresses assigned to a machine" |
| **Step 2.3** | Set the cardinality to Single, since the field will display an integer representing the number of IP-addresses |
| **Step 2.4** | Choose INTEGER as the field format type |
| **Step 2.5** | Select "Element constraint" in the constraint dropdown menu to generate a new Element constraint component that are connected to the field configuration |
| **Step 3** | Select "Computer system" as the asset type in the generated element constraint component |
| **Step 4** | Navigate and select the Rule configuration component in the toolkit and drag it into your workspace. Connect it to the rules socket in the field configuration component |
| **Step 5** | Navigate and select the Field rule component in the toolkit and drag it into your workspace. Connect it to the socket in the rule configuration |
| **Step 6** | Set the constraint to "No constraint" in the field rule |
| **Step 6.1** | Select "Dynamic function" in the function type dropdown menu at the field rule to generate a new dynamic function component |
| **Step 7** | Set the constraint to "No constraint" in the dynamic function component |
| **Step 8** | Select "field value" in the function step dropdown menu in to generate new field value step components |
| **Step 9** | Select "SystemIpAdresses" as property type dropdown menu and then select "Aggregation" in the value step dropdown menu in the field value step componetent to generate a new aggregartion step component |
| **Step 10** | Select "Count" as the aggregation type in the aggregation step component |
| **Step 11** | (Optional) validate the field in the debugger by choosing a asset type of "Computer system" as test object and inspect the returned values |
| **Step 12** | Save the field by clicking on the save action button |

## Creation of an advanced IP-address field

*Background & Task*
This scenario describes a construction process for a comprehensive field that utilized both static functions for static non-inventoried values, relationships in conjunction with fetching values from another field and optimization for retrieving resource type specific field property values. See tables 3.9 and 3.10 for details.

The task is to assign the servers `EsxiHost16` and `EsxiHost30` the static ip-adresses `192.168.11.16` respectively `192.168.11.30`. A potential reason could be that there is a firewall that blocks the application from reaching these servers and the user needs their IP-addresses to be shown in the application. For all other machines we want to show the IPv4 and IPv6-addresses using the field created in a previous scenario. Lastly we want to get virtualization host IP-addresses from HyperV and VMware assets. The finished field should look like the field illustrated in figure 3.14.

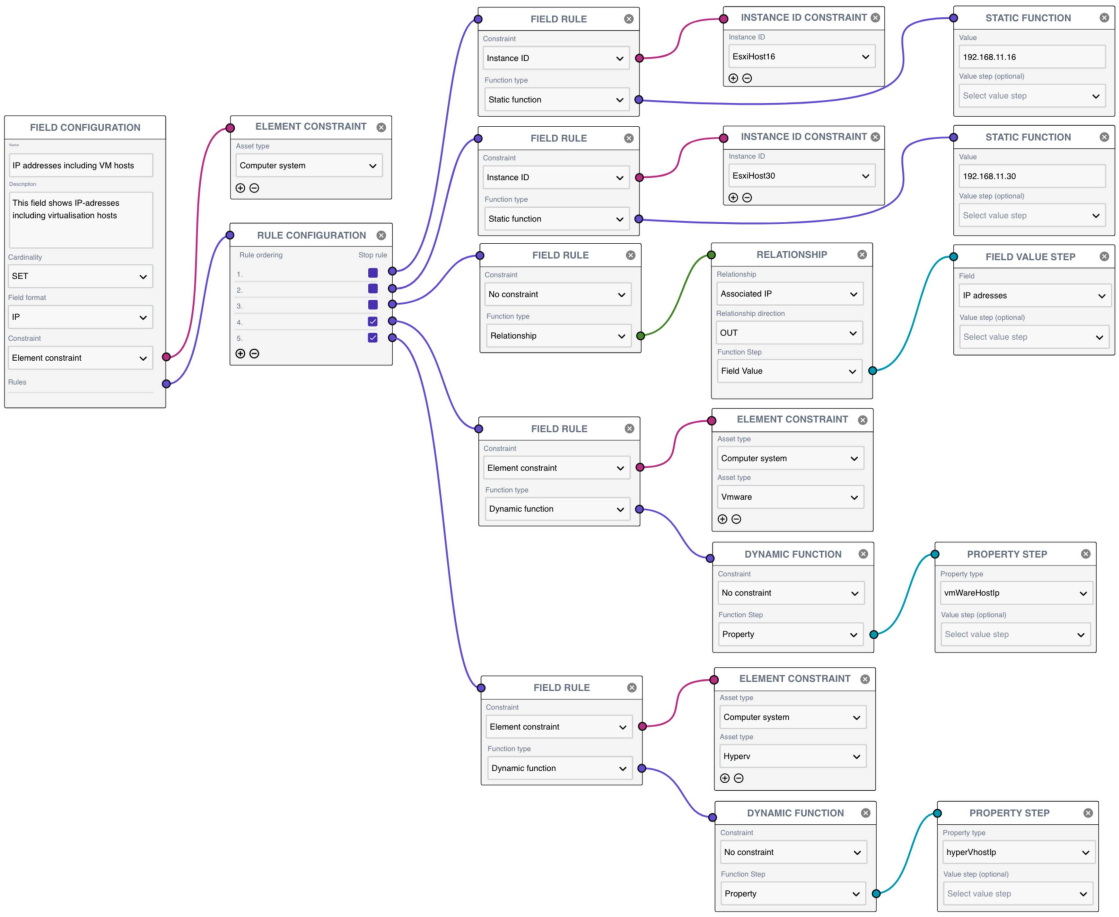



**Figure 3.14:** Implementation of the field in the scenario "Creation of an advanced IP-address field"

**Table 3.9:** Creation of an advanced IP-address field

| **Scenario 3** | |
|---|---|
| Step 1 | Navigate to the Field Builder interface |
| Step 2 | Populate the already existing field configuration component in the workspace |
| Step 2.1 | Choose a suitable name for the field, preferably "IP addresses including virtualization hosts" and type it in the name input field |
| Step 2.2 | Add an description, for example "This field shows IP-addresses for both the organizations servers and virtualization hosts" |
| Step 2.3 | Set the *cardinality* to SET, since machines can have multiple assigned IP-addresses, but only unique addresses are of interest |
| Step 2.4 | Choose IP as the *field format* type |
| Step 2.5 | Select "Element Constraint" in the constraint drop-down menu to generate a new *Element Constraint* component that is connected to the Field Configuration component |
| Step 3 | Select "Computer System" as the *asset type* in the generated element constraint component |
| Step 4 | Navigate and select the Rule configuration component in the toolkit and drag it into your workspace. Connect it to the rules socket in the field configuration component |
| Step 5 | Navigate and select the Field rule component in the toolkit and drag it into your workspace. Connect it to the top socket in the rule configuration by clicking and dragging from either socket of a component |
| Step 5.1 | Set the constraint to "Instance ID" in the field rule to generate a new Instance ID Constraint component |
| Step 5.2 | Set the Instance ID property in the Instance ID Constraint component to `EsxiHost16` |
| Step 5.3 | Select *Static Function* as the function type in the Field Rule component to generate a new static function component |
| Step 5.4 | Type `192.168.11.16` into the newly generated Static Function component |
| Step 6 | Repeat the procedure from step 5 but use `EsxiHost30` as instance ID constraint and `192.168.11.30` as static value |
| Step 7 | Click on the plus icon on the Rule Configuration component to create a new function socket. Then drag a field rule component from the toolkit and connect it to the socket |
| Step 7.1 | Choose "No Constraint" as constraint in the new Field Rule |
| Step 7.2 | Select *Relationship* in the function type to generate a new Relationship component |

**Table 3.10:** Creation of an advanced IP-address field

| Scenario 3 (continued) | |
|---|---|
| Step 8 | Select the Associated IP relation in the relationship drop-down menu, set the relationship direction to OUT and finally field value as function step to generate a new field value step component (The relationship is used to fetch an IP resource type from the scope of computer systems) |
| Step 8.1 | Choose the field IP addresses in the field drop-down menu. |
| Step 9 | Click on the plus icon on the rule configuration component to create a new function socket, Then drag a field rule component from the toolkit and connect it to the rule configuration socket. |
| Step 9.1 | Select the previously connected field rule as a stop rule in the rule configuration component |
| Step 9.2 | Choose "Element constraint" as constraint in the new field rule to generate a new element constraint component |
| Step 9.3 | Click on the plus icon in the element constraint component to create an additional constraint, populate both constraint with `computersystem` and `VMware` |
| Step 9.4 | Create a new dynamic function from the connected field rule and populate it with no constraint and choose property as function step to generate a new property step component |
| Step 9.5 | Choose `vmWareHostIp` as the property type in the property step component |
| Step 10 | Repeat the actions in step 9 with the values `HyperV` and `hyperVhostIp` |
| Step 11 | (Optional) validate the field in the debugger by choosing a asset type of "Computer system" as test object and inspect the returned values |
| Step 12 | Save the field by clicking on the save |

### 3.4.3 Evaluation of Lo-Fi prototype

Four users participated in the testing of the prototype. Participants were all male, between the ages of 25 to 35. Each session had a duration of one hour, divided into a briefing, execution of a scenario and a debriefing where a discussion about user's experience of the prototype took place. As all of the participants were aware of our work and had a degree of familiarity with underlying concepts of the Field Builder, they did not require an extended introduction. As test scenarios for the prototype we used the detailed prototype scenarios described in previous section.

After conducting testing on the user groups with lower technical expertise - content manager and product owner we observed that even for users that had a high degree of familiarity with the product, the initial exposure to the construction process was overwhelming. These users had a high level understanding of what a field represents, though they lacked the understanding of how a field functions from a programmatic perspective. Due to their experience with the product, we anticipated that the task presented would not be as challenging as it turned out to be. They understood the goal of the scenario we asked them to execute, but they were quickly confused by the terminology, such as what a "step", "constraint" or "dynamic function" nodes represented. This outcome was somewhat expected, as we had only performed slight modifications to the terms that were used.

The conclusion was that the challenge of creating an abstraction layer in the form of more user-friendly terminology was greater than we originally anticipated.

Another issue arose concerning the dichotomy of input and output to/from the database. It was not clear that a static value in a "static function" is something that a user defines and is inserted into the database. This is a good example of terminology that needs to be reformulated to better convey the functionality that is presented.

During a discussion after one of the testing sessions a user expressed that if he had been shown beforehand how a field is constructed using the Field Builder interface that he would feel much more comfortable executing a task. The point he brought up can be related to one of Nielsen's Usability Heuristics [42] - "# 6: Recognition rather than recall". Based on cognitive science and it's current understanding of how memory works, Budiu argues that recognition is preferred over recall, as it involves more mental cues which affects how easy it is to retrieve a specific memory [43]. Recognition is related to an understanding of what something does, while recall is related to prior experience with the topic in question. Recognition can be facilitated by easily accessible presentation of available interface functions.

Ideally, in a more developed version of the Field Builder, a user would have the option to view one of the available fields in the workspace and gain an understanding of how a field is typically constructed. A user may initially rely more on recall than recognition as he or she becomes familiar with the pre-made field configurations. Subsequently, as the user interacts and gets more familiar with the Field Builder, recognition will become the more prevalent approach. Hypothetically, the higher the degree of technical knowledge a user possesses, the more he or she would utilize recognition over recall. Having an understanding of the inner workings of a field configuration is advantageous, as the user would primarily need to recognize the terminology that is used, rather than having to remember how to connect various nodes.

From our interviews we gathered that it was common for the tag manager to be used primarily in the initial phases after a purchase. A typical intention was to create all the initial tags that would be of relevance to the customer and using them as key performance indicators accessed as widgets in dashboards. This fact emphasizes the value of striving towards a high degree of recognition to enable users to get started quickly.

During the prototyping we consulted two back-end developers to receive feedback about an iteration of the interface by showing them a couple of screenshots of the work in progress. The primary purpose was to ensure that we understood the technicalities of the process for constructing field configurations, before continuing the development of the prototype. The feedback was positive, with some general comments about the workflow. At a later point during the design process we decided to exclude the developers in question from future testing of that particular iteration. This was motivated due to their exposure to the work in progress as they would have somewhat of an advantage coming into the testing procedure which would influence their performance.

During testing, a user from the product owner group expressed a desire for a clear anchor point in the field configuration. A typical question a user might have is "What do I have and what do I need to do next?". The purpose of such a function would be to provide the user with a starting point and a sense of direction in which the user is to proceed.

To address this concern, the field configuration root component can not be moved or removed. This constraint was added to solidify it's status as the central, mandatory node.

Additionally it has a unique color to highlight it's special status. However, for this prototype iteration and due to its low fidelity nature it was not clear that the functionality in question was present in the field configuration root node. The only hint that this root node is a starting point for the configuration is its name and that it contains only input sockets.

To further aid the user, an additional functionality was proposed. Upon selection of an alternative from a drop-down menu that implies an addition of further components, the proper component will be generated in the workspace and connected to the initial one.

Further the user expressed a valid concern regarding the end of the construction process - how or when does a user know that he or she is done with field? There are two aspects - a user needs to know that the field functions correctly and that it works as intended i.e. it returns and/or inputs the desired values from/to the database. This reinforce discussions regarding these points that we had when we performed the flow map design method in phase one. To address this issue two features were planned - validation and evaluation of the field configuration. See section Reflections & remarks about features for further discussion about these features.

## Creation Wizard

During user testing in the Lo-Fi prototyping phase a discussion was held regarding the initial steps in the procedure of creation of a new field configuration. The initial steps of the construction process are of great significance concerning new users. The creation wizard is a proposed solution for an introductory tool. See figure 3.15 for a prototype created in Adobe XD. The wizard provides a step-by-step process of setting up the initial parameters and properties of a field or relationship configuration, with brief descriptions for each of them. The interface of the wizard is structured as a step by step procedure, allowing the user to go back and forth between the different set-up steps. Aside from the name and description of the field, every other choice is selected from drop-down menus.

Each configurable parameter has a short description that accompanies it and explains the meaning of each particular setting and it's relation to the functionality of the configuration. The wizard is thus a fitting place to include details about terminology in use and various implementation details. One drawback of the current iteration of the prototype is the lack of informative features, such as tool-tips, that would fill the role of an informative output from the interface.

Once the user is finished with all the steps in the wizard a schema reflecting the choices made is presented in the working area. Nodes are placed in a structurally compact pattern, with their interconnections in place. For some of the simplest use cases the user needs to enter a value into a node or add just one or two additional nodes to complete the configuration. For example, in the case of user wanting to set an own-defined (static) value for a specific asset. The wizard would, upon completion, present the user with a complete schema and the only remaining input required from the user is to enter the desired value into the field inside the last node.

In consideration for the eventually more experienced users of the Field Builder, an option to skip the wizard and start with an empty configuration was deemed to be useful. This was done to avoid forcing the user into a certain workflow and give a freedom of choice in the process. Once a user grows more accustomed to the workflow, the need for explanatory aspects in the wizard decreases and the user can configure all desired proper-

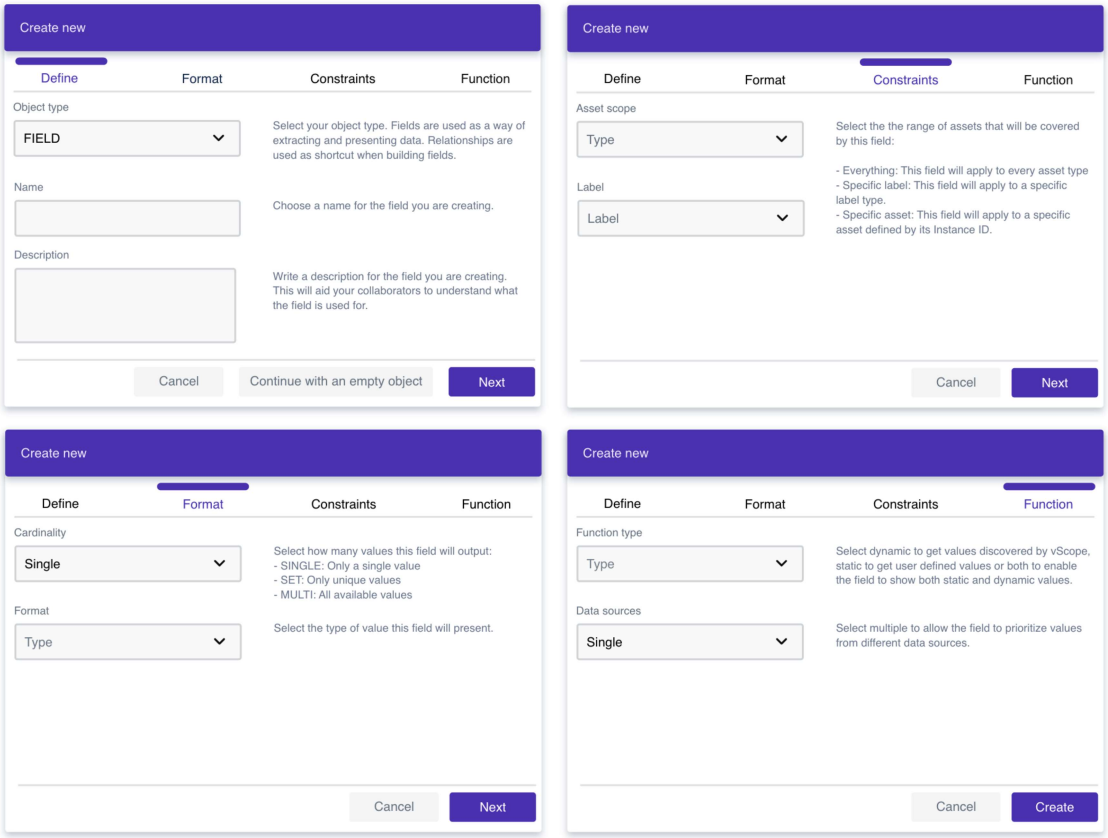



**Figure 3.15:** Creation Wizard

ties directly in the schema from the beginning. For example, in a case when a user wants to create a field configuration that is a slight variation of an already existing one, he or she might open it in a separate tab to use as reference. Another solution to facilitate this scenario would be to offer the user with an option to select an existing field, open it in the workspace and save it as a separate, new field.

In later phases of the project, due to time constraints, the decision was made to omit implementation of the wizard in the scope of this thesis, as it is a feature that is primarily aimed at users with little to no technical expertise in the database domain. The feature was well received by our supervisor and other team members and is planned to be implemented in a future version of the Field Builder.

### 3.4.4 Reflections & remarks about features

This section is intended to act as an inspiration for future work and the potential development of a fully working system.

#### Reorganize & Formatting

One of potential drawbacks with visual programming is the risk for visual cluttering when the number of elements increases in the interface. In the case of the interface developed in this thesis there is a possibility that the fields become functionally complex and thus

visually large. A large graphical schema may result in long or overlapping connections that decreases the overview and in turn readability of the logic that builds up the field.
In most of the common IDEs for text-based programming languages it exists functions for formatting and structuring the code in a unified way, for example introducing line breaks for code snippets that span over a certain length, removing unnecessary spaces and setting the indentation corresponding to the scope of the code.
A way to decrease the visual cluttering in the interface could be achieved by leveraging the same structuring principles, but instead the formatting will reorganize the steps to reduce the length of the connections, prevent overlapping connections and place components that are closely related in terms of the number of connections the path between them have. The same functionality could also be used to organize fields that never have been visualized before in case they have been implemented programmatically or auto generated.

## Naming convention & terminology

A recurring point of discussion during several of the design methods involving users, such as Lo-Fi testing and card sorting, was the naming convention and terminology presented to the user. A complicated terminology can feel overwhelming when a user is exposed to an interface for the first time. One of the main challenges in this work was defining and applying an abstraction layer between the structure of the graph database and the user in a way that bridges a user's mental model with the conceptual model of the system.

The difficulty of this task is present mainly due to the database, which the service runs on, being developed in-house thus a lot of the terminology describing various features and elements is defined by the developer team and is unique to the product. Many of the terms are vaguely descriptive from a general user perspective, and thus not very useful when transferred directly into the interface. To ease in a user into the process, a creation wizard was conceptualized and created as a prototype.

## Graphical Schema

The structural layout of the graphical schema inside the workspace can grow horizontally and vertically, depending on the complexity of the field. Each additional node beyond the root configuration node can be logically placed into a row and column, although the working area allows for arbitrary placement of each node anywhere within it without any constraints. A node can be placed anywhere in the two-dimensional space, the only relevant aspect for the functionality of the field is the interconnections between each node. Generally the schema grows more in vertically than horizontally with increasing complexity of the field configuration. This growth in the vertical axis is mainly due to complexity being proportional to the number of rules that are included in the configuration.

## Graph exploration & debugging

The creation process of fields today often involves an exploration phase where developers navigate and explore the graph's internal structure. For instance, if we take the example relation from chapter two, the developers needs to know the relation between a `computersystem`, an `ipaddress` and particularly the `BindsToLANEndpoint` node

in between, in order to create the relation. This is often done by exploring the graph by using Gremlin commands in a console interface to the graph, where they can use a known vertex ID as a starting point and then traverse the graph by looking up the the available relations and properties of the vertices. This process is quite cumbersome for developers not fluent in Gremlin and the console interface lacks good structuring due to all properties and relations getting listed in a single line output separated by commas.

With this background established, a feature that supports this workflow could be motivated in the Field Builder interface in form of an debugging tool. We propose that the design of this feature should mimic the existing work flow used by developers by letting the user choose an existing asset to use a test element. When a traversal algorithm is created it is generally applied on all assets specified by the constraint property, but in the case of using it as an test element the scope of the traversal algorithm will be modified to only evaluate the test element rather than all elements of that type. Thus providing recognition and feedback for the users.

As the task analysis revealed, users often have a high degree of familiarity with their own data set. By showing the evaluated vertex properties it will possibly aid them in the process of constructing fields and act as insight into the otherwise black-box model of the database.

In our Lo-Fi prototype we implemented this feature with only one interaction step, and that was the possibility to open the sliding view by clicking on the arrow in the upper right corner. The primary reason to include this basic interaction in the prototype was to let it act as an foundation for the discussion after the testing session and to evaluate if the test subjects would understand it's usage and how well it corresponds to their mental model.

The interaction sequence for this feature is imagined to progress by the user first selecting a test element by clicking on the icon in the upper input field. Then, a prompt pops up as an overlay over the workspace where the user is asked to choose an asset type followed by selecting the specific asset. The user then selects a node in the workspace where the field is currently being constructed and it becomes highlighted in terms of the border of the node becoming purple just as the evaluate button. The user then clicks evaluate in the bottom of the debugger view and the properties and relations of the vertex get listed and evaluated, while the list is processing a loading icon will be displayed in the first row of each table. If the selected node generates a field, properties of the field get listed. If the node generates field values, they get listed as they would be shown in the table explorer.

An illustration of the debugger view is presented in the appendix, figure 5.

## Context Aware Filtering

Content aware filtering is a method for enforcing constraints on the interface by reducing the number of available options to the user. These options include the nodes themselves and their properties. This filtering is applied on the basis of that not every existing node or property makes sense in the context of the current state of the field configuration e.g. *useraccounts* can not have a *RAM* property assigned to them. Filtering is done in real time and is based on user input. Due to the complex nature of the domain space of this application the purpose of this feature is to mitigate the complexity by removing the ability to select invalid options. In addition to guiding the user through the process of construction, content aware filtering acts as a preventive measure against faulty field configurations.

See the section on Validation / Evaluation for discussion about these aspects specifically. With regards to developers it might be desirable for this feature to be optional, to allow for experimentation with new steps without getting hindered by the filter. An important aspect to take into consideration when designing this feature is the visual change of state of the interface. It should be made clear to the user why the interface changes upon certain actions, and for what purpose [41].

The primary value of content aware filtering is that of reducing the cognitive load by limiting the number of available nodes the user has access to in a particular state of the graphical schema. Nodes that are not suitable will change their visual state to reflect their incompatibility with the current state of the schema. One way of implementing this would be to reduce the opacity and applying a grayscale overlay to unsuitable nodes in the toolkit. In the case for incompatible properties within nodes, they would be hidden. When the user clicks on a node in the workspace, content aware filtering will interpret the user action and propagate a state change to the toolkit. For instance, if the user selects the root "Field Configuration" node, it will act as the primary context within which the filtering will apply. Consequently, every node but the "Element Constraint" and "Rule Configuration" will become grayed out in the toolkit. For example, in Figure 3.12, when the user has added the "Dynamic Function" node to the workspace and selected "Field Value" in the second drop-down menu, only the "Field Value Step" node in the toolkit would retain its visual state, while every other node would become grayed out. After the user drags the "Field Value Step" node into the workspace, the same procedure will apply to "Aggregation Step" node instead, as it will register as the last selected node.

## Traversal State Overview

From our user testing of the Lo-Fi prototype, the users experienced difficulty in understanding what the different steps had as input and output elements in terms of the vertices, fields and field values. The color coding on the edges and sockets that we applied with the intention to make the relationship of the input and output more intuitive were not as obvious for the users at their first glance as anticipated. However, it all made much sense when we informed them of the meaning of the color coding.

An additional design feature to help users with identifying the appropriate input/output element type is a proposed consideration for future work. As a potential candidate for evaluation for this feature we propose the addition of an icon to the node. The icon would correspond to the type of input/output element of the node.

The discussion that followed after the testing session revealed that the concept of vertices, fields and field values was not as clear for the non-developer test subjects even with their background of working close to the technical aspects of the product. From this insight it is clear that this aspect of the design needs to be improved with more intuitive signifiers in the next coming design iteration.

## Field validation/evaluation

Validation and evaluation are the last barriers of entry before a field configuration is saved to the database.

Validation is an additional safety layer, alongside content aware filtering, to ensure that a field configuration is compatible with the database and can be executed by the engine. While content aware filtering is relevant during the process of construction of the field configuration, validation is performed at the end of the process. Validation can be compared to a syntactic check of an IDE or a compiler, which warns the user in case there is an error in the syntax of the program.

Evaluation is the process of examining the output of a field configuration and is performed through the debug functionality. The evaluation functionality was primarily introduced as a way to let the users evaluate the field in the context of their own data set, which assists the user to see if the field behaves as intended. This is useful in cases where the user may have misinterpreted or misused some of the steps. The usefulness of this functionality depends on the degree of familiarity a user has with his or hers data set. It also serves as a debugging tool allowing the user to follow the execution trace to find anomalies in the algorithm.

## Work-in-progress states and tabs

The insights from the task analysis motivates the usage of tabs in the interface, thus enabling the user to switch between different fields implementations to act as inspiration when creating fields and support parallel work flows.

Every tab will act as a stand-alone session that stores the current state of the field. This allow users to continue their previous work if they have navigated to a different view within the application or had a context switch in their work flow.

Another reason to introduce a work-in-progress feature in the interface is to enable the user to temporarily store invalid fields, as the save action is protected by a validity check to prevent the user from bringing faulty fields into the application.

## Locked fields & clone feature

As described in the section for the technical implementation for fields, the fields contains a attribute `fieldType` that tells whether the field is created by a user or comes pre-bundled with the application. The company ensures that the pre-bundled fields displays data with high quality and thus it's important that these fields remains unmodified by users. The only way to edit bundled fields is by software updates provided by the company.

There exists several reasons for making these fields visible for the end users. Firstly, to give the user the possibility to look inside a bundled field to get an understanding of how a particular field is constructed and what values the field is expected to return in case the description is not enough. Secondly, make them available as an inspiration source for the users when they construct their own fields. Thus, a clone feature could be motivated so the users can clone a bundled field and use it as a template for their own field.

A potential way to inform the user that the field is bundled and can not be modified could be to introduce a signifier in form of an padlock in the list element of fields in the field browser.

## Regular expression based filter

Both the task analysis and the data-driven user analytics method brought up a problematic area regarding the filtering associated with the current tags. Many use cases with static values could be solved with the current regex feature, but often the users lacked knowledge about regex. In some cases when users specifically asked if the filtering could be achieved with regex, the workshop facilitator was not aware of the feature. This is a clear indicator of a shortage in the design of the application. Although, the feature is described on the product's support portal, its implementation is not well suited for a large portion of the user base.
The example presented in the support portal is
`regex:(?sui).*sql(?!.*express).*` and it's applied on the guest database software tag. It is described by a scenario, where the user wants to include all machines that includes sql and excludes express in the mention tag value. Obviously, this approach expects a extensive knowledge of regex, which often is not the case for the majority of the user base. However, it exists one abstraction of the feature and it's the case when the user just wants to filter on a tag that includes one specific sequence of characters, for example `*sql*` will filter on all values that includes the word sql.

From the workshops we attended as observers, we noticed that many of the questions from the customers related to filtering shared some common characteristics. Often the customers wanted to create new tags that showed a specific value based on other user-defined static tag values. An example were when a customer wanted to map server descriptions to a locations. They had a naming convention within their organization for the name of their servers, for instance `MO-122-T`, `MO-134-P` could be servers located in Malmö and `GBG-102-P`, `GBG-133-T` in Gothenburg, where the T and P stands for whether the server is used for testing or production purposes. Example use cases could be to map all servers beginning with `GBG` to a static value `Göteborg`, `-T` to `test server` or values that includes both `MO` and `-P` to `production server malmö`.

With the new release of the application filter functionality will be placed inside the Field Builder due to it's close relation to fields. The step for this function is currently not implemented but we assume it will be implemented with a constraint step that has an field value step as input. We propose that a potential representation of this step could be divided into two different components in the interface, see figure 3.17 for an illustration. One of the components would consist out of an input field that accepts plain regex to allow both power users and developers to express themselves in a format that they are comfortable with. The second component will be targeted towards the average user without any prior knowledge of regex. This component consists out of two sections with input fields to target a specific sequence of characters to include or exclude. A tool tip over the input fields displays the possibility to include a star to target a arbitrary character.

## Toolkit and browse functionality

The *browse* and *toolkit* menus overlapped to a degree in their semantic meaning, in that both components offer a way to browse available fields and relationships, thus working with the same type of objects. The difference between them being that the browse component is used to open the context of a field or relationship in the working area, while the toolkit is used to allow the usage of an existing field or relationship in the currently open

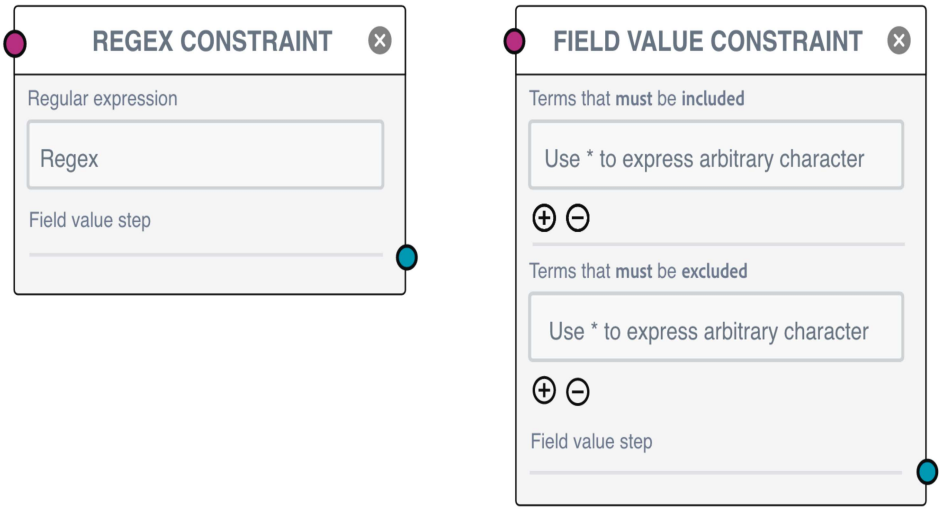



**Figure 3.16:** Regex constraint & Field value constraint

context as an element in the current configuration context.

A solution through the usage of a single panel encapsulating both menus by adapting a right click context menu to aggregate the functionality was evaluated, however it was found to be obscure and non-intuitive. It was not clear at a glance what functionality was offered by the presented menu.

We opted for an implementation of two separate menus. When the user initially opens the Field Builder, he or she is presented with the *browse* menu and an empty working area. At the top of the menu is a *Create New* button which changes the "*browse*" menu to the "*toolkit*" menu. This approach proved to provide a more clear state of the process which the user currently was at.

### Toolkit Layout

For the placement of the toolkit, the UI element containing available nodes, two options were considered. The first alternative was to place the toolkit on the left or right side of the working area, and the other is to place it at the bottom. Placement along the top side of the area was quickly disregarded as it was too obtrusive covering a large part of where the working area would be.

After conducting user testing on Lo-Fi prototypes the placement of the toolkit on the left side, alongside the primary menu of the service, was deemed to be most fitting and non obtrusive. Additionally it made sense for all the menus in the interface to be located in the same area. Se figure 3.10 for reference.

## 3.5 Phase four

In the fourth phase of the design process, we focused on converting the Lo-Fi prototype into a Hi-Fi prototype inside the in-development version of the product. It was then evaluated in

regards to the goals and requirements of the project. A large part of this phase involved us getting familiar with the development environment, development language and the library that we used for implementation.

### 3.5.1 Hi-Fi Prototype

For an implementation of the Hi-Fi prototype we used JavaScript React to allow more advanced interaction workflows than those offered by Adobe XD, such as rendering responsive connections between nodes and the ability to move and zoom around in the workspace. Due JavaScript's programmatic nature it is possible to enforce logical constraints on compatibility between sockets. Further, it is possible for users to approach the construction process in an unrestrained way, not having to follow a predefined sequence of actions.

Based on the feedback received from the Lo-Fi prototype we applied two changes to the user interface. The first change was aimed to address the starting point matter that one user commented upon. As the root field configuration node is an essential part of the traversal algorithm, we configured it to be present by default within the workspace whenever the user begins the construction of a new field. Additionally, it is not possible to move it around the workspace. Consequently, we removed the field configuration node from the list of available nodes in the toolkit, as there is no point in having the option to add more than one root node at a time in the workspace.

The toolkit functionality was implemented in a bare-bone way, compared to the design presented in the Lo-Fi prototype. The primary goal for this iteration was to evaluate the construction workflow. In its current state, the toolkit only offers various nodes that can be added to the workspace by dragging and dropping them from the toolkit. Nodes are listed in the toolkit ordered by their type. As an additional way to visually differentiate between various types of nodes we added color coding to their headers. Only nodes that are relevant for the execution of test scenarios were included. Figure 3.18 shows an overview of the final Hi-Fi prototype.

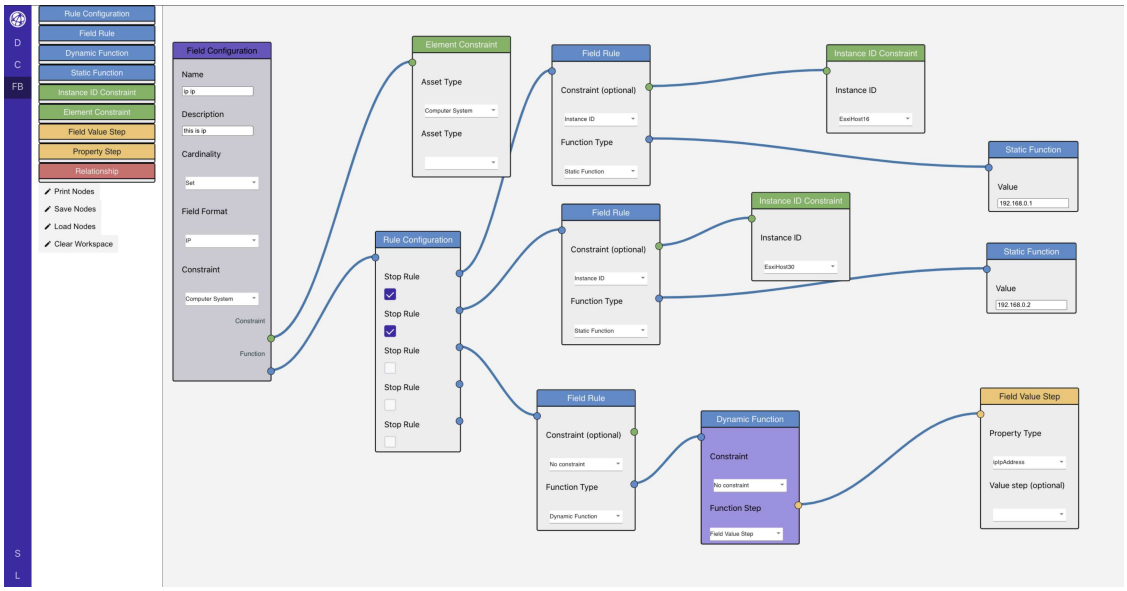



**Figure 3.17:** Hi-Fi prototype

In the final iteration of the Hi-Fi prototype we implemented a context menu functionality accessed by right-clicking on a node within the workspace. This menu contained two options - delete and clone. The delete option, when clicked would result in removal of the node from the workspace. Any present connections between the removed node and its siblings are removed as well. The context menu was disabled for the root configuration node as none of the actions are applicable to it, due to the constraint on the presence of exactly one root node within a field configuration. The inclusion of the delete functionality in the context menu was a temporary solution for this version of the prototype. The main drawback with having delete functionality in the context menu is that it is hidden behind an invisible affordance. Looking ahead it would instead be accessed through a "circled cross" icon located at the top right corner of a node, illustrated in the detailed prototype scenarios. This was the preferred solution based on the feedback received from the Lo-Fi prototype. The clone option creates a copy of the selected node. This was option was implemented for cases when the user wants to quickly add multiple nodes of the same type, as is common for nodes of type "Static Function", "Element Constraint" or "Field Rule". This streamlines the construction process by removing the need to find a node in the toolkit once it has been added to the workspace.

### 3.5.2 Evaluation of Hi-Fi prototype

To evaluate the Hi-Fi prototype we performed user testing with one of the developers from the company. Due to demanding workload towards the end of the quarter, the availability of developers was very low, thus we unfortunately were able to involve just one person. On the upside, the chosen developer was representative of the general level of programming knowledge and technical understanding of other developers employed at the company. The testing was conducted remotely with the usage of screen sharing software. During testing we wrote down notes and talking points for the discussion that took place after a scenario.

The test contained two scenarios based on scenarios from section 3.4.2. The first scenario was "Creation of an IP-address field", with it's primary purpose being to introduce the user to the interface. The second scenario was "Creation of an advanced IP-address field", requiring inclusion of an increased number, as well as different types, of nodes compared to the first one to fully implement a working field configuration.

The user got briefed with the description and goal of the scenario he had to complete. The briefing was comprised of the background information to define the context within which the field would be applied. At the start of each scenario the user was presented with an empty workspace, with the exception of the root node. In both scenarios the user applied an interesting approach which resulted in new insights in regards to the construction process. These insights motivated an application of certain workflow optimizations for future versions of the Field Builder. As seen in the scenarios there is a number of repeating steps, especially those covering connection of components. The number of required user actions could be greatly reduced by a context aware functionality. For example when a user adds more than one *Field Rule* node, the *Rule Configuration* node would automatically be inserted into the workplace. Each *Field Rule* node would accordingly be connected to a socket in the *Rule Configuration* node. Further, the *Rule Configuration* node be connected to the *Field Configuration* node as the *Rule* property.

When we inquired about the terminology in use, he expressed that he felt comfortable

due to consistency of terms between his usual working environment and the Field Builder interface. He added that there could be a point to abstracting certain aspects, particularly with regards to new developers at the company. For the most part, however, he felt that too heavy abstraction could be a hindrance as it could reduce recognition.

In the discussion after the scenario he explained that the inclusion of a *Property Step* implies the presence of dynamic function that the step node must to be connected to, as it is the only compatible output node. See his solution in figure 3.16 as reference. This led to discussion and formulation of a feature that would allow users to omit the inclusion of certain nodes. Once the user is done with the construction of a field configuration, the graphical schema would be interpreted and if any implicit nodes are detected, they would be injected into the back-end API request.

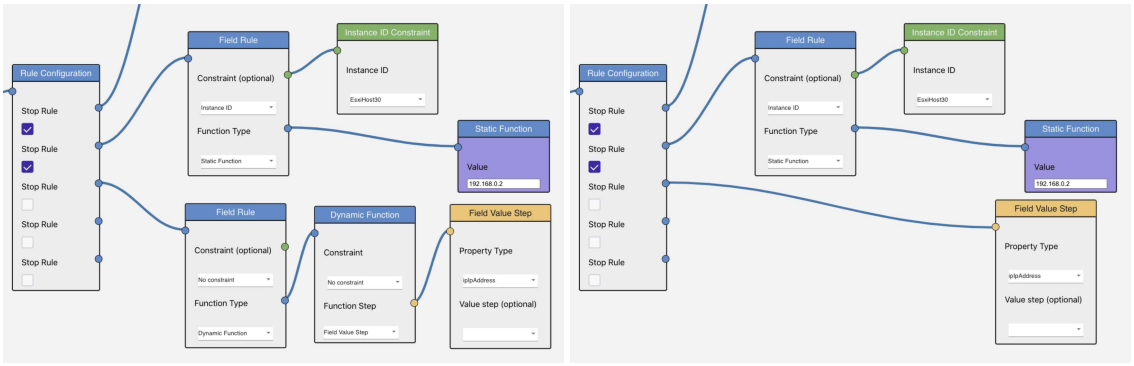



**Figure 3.18:** Functionally equivalent constructions

### 3.5.3 Heuristic Evaluation

The heuristic evaluation design method was chosen to act as an complement to the already performed user evaluations of the Lo-Fi and Hi-Fi prototypes, due to a very limited amount of test subjects. There exists a potential risk that we introduce bias into the conclusions of this evaluation since we are the designers behind the evaluated user interface. It is also recommended to use usability experts and preferably ones with domain expertise. Bailey et al. states that heuristic evaluations often introduce *false alarms*, which is when the evaluators identify usability problems that do not seem to actually be problematic for the end users [44]. For these reasons, the motivation behind performing this method is debatable, but the conclusions could be used as a starting point for future work where potential usability problems with the current prototype get exposed.

We conducted the evaluation individually by studying the interface with Nielsen's ten user interface design heuristics in mind. We wrote down our thoughts regarding the heuristics to later discuss them together and formulate a compiled list of the conclusions, presented below. We could expect to identify roughly 50% of usability related problems by conducting the evaluation in two individual instances according to Nielsen [45].

#### 1. Visibility of system status

This iteration of the prototype does not communicate with the back-end of the application. All actions are performed client-side. Hence, the system is always interactable.

However, when this prototype matures into a version ready for deployment in production, the following operations must provide clear feedback for the user regarding the system status.

- When the system load fields or relationships to be displayed in the workspace.
- When the lists in the toolkit gets populated in terms of available components.
- When the context aware filtering is executing.
- When an evaluation or validation is performed.
- On the save action of a field or relationship.

Most of these features are not particularly demanding for the system but depending on the server load it's favorable to provide some sort of feedback in case the operations requires some time to execute.

We propose that a loading spinner could be implemented and shown on the element causing the system status to be changed. In case of the save and evaluation action, a loading spinner could replace the text present on the button, while the action is executing. The same principle could be applied on the list in the toolkit when operations that change it's content is being executed. And when the user loads a field or relationship into the workspace a loading spinner could be placed at the center of the workspace.

## 2. Match between system and the real world

As the scope of implementation was narrowed to focus on developers, the information presented and terminology in use mirrors the environment they are accustomed to. Thus, at this stage it is sufficiently satisfied. However, looking forward to future development of the Field Builder and as the user base expands to target other types of users, this heuristic requires arguably the greatest amount of effort to properly apply.

## 3. User control and freedom

One way of providing user with greater control of the workflow is to ensure the presence of common functions such as undo, redo, copy and paste actions. Typically these actions are available through keyboard shortcuts as well as icons in the interface. Current prototype iteration is lacking in this regard, as it does not offer an undo/redo workflow, and only a limited variant of copy/paste. While the absence of undo/redo functionality is not a critical hindrance, it is a common feature that most users expect to be present within an interface where one works in a sequential manner. It is possible to perform a "copy and paste" action on a node by using the "Clone" option from the right-click context menu. Ideally it should be available through keyboard shortcuts as well. The reason for these shortcomings in this particular heuristic is that they had low priority in this prototyping phase in relation to more fundamental features.

## 4. Consistency and standards

This heuristic centers around cohesion in the interface by ensuring that graphic elements and terminology is consistent throughout the design. With regards to visual aspects of the interface, this heuristic was initially utilized in the Lo-Fi prototyping phase when we used Adobe XD to ensure compliance with design guidelines established by the company.

## 5. Error prevention

One aspect in the prototype that reinforces this heuristic is achieved through the usage of drop-down menus for selection of properties in a node. By limiting the number of options a user can select, the potential for errors is reduced. This is applied to every node, with the exception of the *Name* and *Description* properties in the Field Configuration node and the *Value* property in the *Static Value* node. The reason being that these properties can contain arbitrary values.

Further we have enforced constraints on input and output sockets of a node, allowing only logically compatible sockets, divided by type, to be connected to each other. Additionally sockets are color coded to signify their type.

## 6. Recognition rather than recall

As designers of the object for this evaluation, it is challenging to properly do so when it is related to memory, as we are exposed to the interface and its functionality on a daily basis. However, this heuristic is given special consideration in the general section on Lo-Fi prototype. Ensuring that heuristics #2 and #4 are employed, will provide support for this one as well, as consistency and coherence are important factors for recognition.

## 7. Flexibility and efficiency of use

This heuristic relates to the availability of function keys, shortcuts and interface customization. As these aspects were not prioritized for implementation in this iteration of the prototype, it is not suitable for an evaluation.

## 8. Aesthetic and minimalist design

For future work, the Lo-Fi prototype created in Adobe XD should be used as the primary reference in regards to the visual design. The reason being that the design reflected is more coherent with design guidelines of the application because visual aspects had higher priority in the Lo-Fi prototyping phase. The Hi-Fi prototype was primarily focused on functionality and facilitating evaluation of the workflow. Every element presented in the interface has a specific purpose, thus a degree of minimalism has been achieved.

## 9. Help users recognize, diagnose and recover from errors

Error diagnostics is an important aspect when a user is working within a complex domain space. This heuristic was not applicable for evaluation for this prototype iteration. As a starting point, this heuristic could be satisfied for by implementing simple error messages

that briefly explain erroneous user actions. With regards to this version of the Hi-Fi prototype such messages could be displayed when a user attempts to connect incomaptible socket types. Other diagnostic features, evaluation and validation of fields are presented and discussed in the section about the Lo-Fi prototype. We propose that they could be further enhanced by providing users with a tool for visual overview of the graph database structure, where an evaluation of traversal algorithm would highlight all affected vertices.

## 10. Help and documentation

A design should strive towards a level of usability which minimizes a user's need to resort to documentation, but depending on the solution there might still be a need for one. The company offers a support portal for the application which has pages describing some basic features and information about commonly request support cases. A documentation for getting started with the Field Builder could potentially be included in the portal. The scope of this thesis does not cover compilation of a documentation, however it is an aspect that should be taken into consideration when a Field Builder prototype has reached a sufficient level of maturity. An indirect way for a user to receive guidelines for creation of a field is to open an already existing one in the Field Builder an analyse its structure.

# Chapter 4
# Discussion

This chapter contains multiple sections that contain our thoughts and reflections on different aspects of this thesis work.

## 4.1 Design process

This thesis was a valuable learning opportunity, providing not only a hands-on experience working with user-centered design methodologies, but also direct practice working in an agile development environment. The availability of design methods is rich in the user design space, with a multitude of approaches applicable for a wide range of problems. Adapting a structured design methodology consisting of four distinct phases proved to be an effective and insightful approach. As we progressed through the phases of the project, advantages of involving users in the design process became more prominent, as many users had useful thoughts and perspectives regarding their mental models and needs. Many of the applied methods had a degree of synergy, enabling the use of data gathered in one method to serve as an additional input layer in another.

### 4.1.1 Prototyping approach

An advantage of using a digital design tool like Adobe XD is that the process allows for quick modifications. The only trade-off with using digital design tools is the learning process, which in our case was relatively short, considering that we only needed the basic functionality. Further, the trade-off for time spent on learning the tool was more than made up for as we began iterating over various design features. Another advantage is that by using Adobe XD, we were able to conceptualize and visualize more clearly how the field builder would fit into the rest of the product, as most of the visual components such as colors, fonts, buttons and icons were available for import. However, to implement more complex interaction as the dynamically rendered connections between the sockets

and drag and drop functionality for the components, we had to continue prototyping in JavaScript react and use additional third-party libraries. Due to no prior experience with JavaScript react, a considerable amount of time was spent to familiarize ourselves the framework. Initially the motivation to use JavaScript React was to incorporate the Field Builder interface into the main application. This would leverage the usage of internal APIs to further extend the functionality of the prototype.

After a discussion with our supervisor, with insights gained from user testing and due to time constraints, we opted to narrow the scope of this work to focus primarily on developing the interface for user represented by the developer persona as an in-house tool. With lessons learned from various design methods we concentrated our efforts on providing the base functionality for enabling the creation of a field configuration, keeping the terminology that developers use and are already familiar with. As developers are a group that are expected to use the Field Builder to a great degree they were chosen as the focus group. Focusing on developers as the target group for the remainder of the thesis work would provide a stable base on which further development can take place.

Due to complex nature of the problem domain of graph traversal algorithms paired together with diagrammatic programming it was initially hard to set a proper estimation for the amount of effort required to investigate, develop and evaluate a suitable interface for the task at hand. While the design methods early on in the process pointed in a promising direction, the amount of effort required became more clear in the later stages of the design process, particularly after conducting card sorting and Lo-Fi prototyping. Due to this, a number of planned features had to be re-prioritized and considered for future work instead.

## 4.2 Aspects of the interface

### 4.2.1 Interface Levels

After conducting interviews we were able to deduct three primary groups which the users could be divided into. The main characteristic separating these groups is their technical understanding of the system, as well as general technical knowledge. A proposition for separating the interface into three separate levels for the different user groups - developers, content providers and general users was discussed. The main differentiating factor between these interface levels is terminology, but the workflow is the same.

The "developer" group includes all the developers working at the company that provides the product. Users in this group have all first-hand experience with the database and deep understanding of the implementation of the system, thus familiar with the terminology in use. Due to this, there is no inherent need for abstracting the terminology, as it could potentially have the opposite effect - leading to confusion in areas where a degree of familiarity is already established.

Users in the "content provider" group have a deep understanding of the product and how it functions from a non-technical perspective. In the current version of the product, "tags" which are similar in functionality to the upcoming fields are one type of content that they provide. Content providers have the possibility to communicate with developers to gain insight about how a field configuration is structured, should they need help with a specific case. Due to this, users in this group have low needs for abstractions in the

interface.

The third group "users" have the least understanding of the technical aspects of the product, and thus in the strongest need for a comprehensive abstraction layer. This group has the highest variance in familiarity with the product, as it includes both new and already established customers. Ideally, the Field Builder should have a low learning curve, enabling new users to quickly produce desired fields.

One of the main tasks for future work is to reformulate the developer defined terminology into a more intuitive, user friendly variant that would be comprehensible by personas "CIO" and "IT Technician". This process likely entails rigorous user studies, evaluating multiple sets of various terms to replace the current domain specific terminology.

### 4.2.2 Visual Programming

One of the main goals for this thesis was designing and providing an interface for creation of traversal algorithms. As we cannot expect an end-user to possess a certain level of knowledge about graph databases and traversal algorithms, a level of abstraction needed to be introduced between the user and the language that communicates with the database.

The visual programming paradigm offered several advantages. First, one of the tasks of this thesis is to design an interface with a sufficient level of abstraction for construction of traversal algorithms. Abstraction is one of the inherent qualities of visual programming languages as it is one of the main goals of the paradigm.

One drawback of visual programming is the risk for visual clutter in the interface, which is somewhat correlated to the size of the program, due to increased complexity leading to increased number of elements and their interconnections. This drawback can be alleviated to a certain degree through several means, one of them is by applying color coding to different elements, for example assigning a color to a particular data type or logical component. Another level of differentiation can be applied through variations in the shape of an element, for example by having rounded corners or variation in number of edges for different types of elements, which is an approach some analysed applications applied.

Additionally visual clutter can be further reduced by emphasizing "good practices" to reduce redundancy on the structural level of the program. A useful feature of textual programming languages is the ability to insert user-defined comments into the code. This could be implemented in a visual programming interface by adding the option to add comments to different graphical elements, or have a separate comment element. Ideally though, the design of the interface (as well as the program) should strive to be self-explanatory, conveying a clear meaning at a quick glance.

During one of discussions with our supervisor a concern about scalability was brought forward. In the long term, it would not be feasible for the company to offer workshops on the Field Builder to every new customer, as it would be too resource intensive with regards to the projected customer acquisition rate. This aspect underlines the demands on the interface to have a low learning curve to enable customers to execute simpler use-cases. As fields can have an arbitrary complexity level, workshops should still offer walkthroughs of advanced implementations on request.

### 4.2.3 Workflow

Another discussed aspect was the direction of the node structure in the graphical schema. Specifically this refers to the placement of the nodes in the two dimensional space. Two main options were considered - flow from left to right and vice versa.

The advantage of a left to right flow is that it presents an intuitive starting point for the schema. Since it requires one required, "anchored" root node, - a "Field Configuration" or a "Relationship Configuration" node, the users agreed that having it presented on the left side felt reasonable, since a parallel to the western "left-to-right" reading style can be drawn.

Visualising a field configuration structure as a tree, with the root configuration node at the top and the various elements such as rules, constraints, traversal functions as its branches, expanding down. Using this format a top-down and bottom-up dichotomy for construction of a configuration can be defined. The field configuration node encapsulates all other nodes, see the pseudocode block in chapter 2.3.4. The configuration node represents the top level object named *FieldConfigString*.

Top-down describes the process of creating a configuration starting with populating the highest level element - the root node, and then creating other, lower level elements and chaining connections to the root node. Consequently, a bottom-up approach entails creation of the lowest level elements first and connecting them up the configuration hierarchy. This approach, while possible, is inherently more demanding on the user's knowledge of the graph database functionality.

With these definitions in place a parallel can be established. A top-down approach corresponds to a left-to-right alignment of the nodes, and a bottom-up approach to a right-to-left. While both approaches are feasible in the current version of Field Builder, after completing the initial setup by using the wizard, the user is presented with a top-down structure, having to create and connect the last remaining elements. This approach was chosen due to an potato.

### 4.2.4 Color Coding

All node types have one output socket (with the exception of the root configuration node, which only has input sockets, as it is the highest level node), which has to be connected to an input socket of another node. Further, a node has a predefined amount of zero or more input sockets, except for the "Rule Configuration" node, which contains an input socket for each existing field rule. A socket can be of four different types - function, constraint, relationship and step. Each socket is compatible only with it's own type. This constraint was implemented to reduce the number of possible erroneous configurations that would produce an invalid field. When a user attempts to connect two incompatible sockets an error message with an explanation of the incompatibility is presented.

To aid users in the configuration process, we applied color coding to the different socket types. This was done to signify the compatibility between input and output sockets of each node. Color coding principle improves the degree of clarity and reduces room for user-error. Additionally, most nodes contain their output type in their title e.g. "Property Step" has an output with "Step" socket type and "Dynamic Function" has an output with "Function" socket type. (Citat om color coding?)

It is possible to have disconnected nodes in the working area and in such state they will not be considered as a logical part of the field configuration. This behaviour can be useful as a method for quick experimentation for slight variations in the configuration. For example, a user can have two nodes that have the same type, but different content inside their input fields. One can thus quickly swap which node is connected to the rest of the configuration, and evaluating the result.

Potentially, including a "Clean" button could be a useful feature to include in a future version of the product. Upon interaction it would remove all disconnected nodes in the workspace. This can be of value in cases where a user is content with their field configuration and wishes to save it without any residue from the creation process.

# Chapter 5
# Conclusion

Using a visual, node-based interface for creation of graph traversal algorithms is an application of the diagrammatic programming paradigm with a lot of potential. One of the main advantages is that it provides a visual structure in the form of a graphical schema, which has a high degree of provided overview and clarity. While this approach is one that potentially reduces the required level of programming knowledge and technical understanding, a certain baseline for these qualities is still present. This baseline, however, is highly dependant on which domain visual programming is applied in.

Just as a certain level of theoretical understanding in music theory is required to read even the simplest musical notation, or a circuit diagram in the domain of electrical engineering, implementation of graph traversal algorithms puts requirements on the user to possess a certain baseline of domain knowledge. Where that baseline lies is an interesting research question, relevant for investigation in future work. Related research questions are - where is the intersection point between technical understanding and the abstraction layers required for an optimal user experience for creation of graph traversal algorithms? Consequently, what are the proper metaphors and terminology adaptations for an intuitive interface?

As we recognized during our competitive analysis, visual programming is a topic with ongoing interest from academia and the IT-industry. Since their conception, programming languages have been gradually expanding into higher levels of abstraction, leading to a wider adaptation among amateurs, professionals and academics alike. The visual programming paradigm is a natural branch at the top of the abstraction tree, supporting the weight of an increasing interest and demand for programmers in our society.

# Bibliography


[1] Db-engines.com. 2020. DB-Engines Ranking Per Database Model Category. [online] Available at: [Accessed 17 December 2020].

[2] Siau, K., 1998. A visual object-relationship query language for user–database interaction. Telematics and Informatics, 15(1-2), pp.103-119.

[3] Dahl, S. and Lindqvist, K., 1989. Visual programming as an interface between program and user?. [Proceedings] 1989 IEEE Workshop on Visual Languages,.

[4] Vredenburg, K., Mao, J., Smith, P., Carey, T. *A Survey of User-Centered Design Practice* Design methods, Volume No. 4, Issue No. 1, Minneapolis, Minnesota, United States. 2002. pp. 477,

[5] Novak, J. D. & A. J. Cañas, *The Theory Underlying Concept Maps and How to Construct and Use Them*, Technical Report IHMC CmapTools 2006-01 Rev 01-2008, Florida Institute for Human and Machine Cognition, 2008, available at: http://cmap.ihmc.us/Publications/ ResearchPapers/TheoryUnderlyingConceptMaps.pdf

[6] Hyerle, D., 1996. *Visual Tools For Constructing Knowledge*. Association for Supervision and Curriculum Development.

[7] Rosenfeld, L., 2011. *Search Analytics For Your Site*. Brooklyn, New York: Rosenfeld Media.

[8] Nielsen, J., 1994. Design Of Sunweb: Sun Microsystems' Intranet. [online] Nielsen Norman Group. Available at: https://www.nngroup.com/articles/1994-design-sunweb-sun-microsystems-intranet [Accessed 20 August 2020].

[9] McCloskey, M., 2014. *Task Scenarios For Usability Testing*. [online] Nielsen Norman Group. Available at: https://www.nngroup.com/articles/task-scenarios-usability-testing/[Accessed 18 July 2020].

[10] Kuniavsky, M., 2010. Observing The User Experience. San Francisco, Calif: Morgan Kaufmann, pp.419-428.

[11] Warfel, T., 2010. *Prototyping*. Brooklyn, New York: Rosenfeld Media.

[12] Laubheimer, P., 2020. *3 Persona Types: Lightweight, Qualitative, And Statistical.* [online] Nielsen Norman Group. Available at: [Accessed 15 July 2020].

[13] Nielsen, J. *Finding Usability Problems Through Heuristic Evaluation* Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 1992.

[14] Angles, R. and Gutierrez, C., 2008. *Survey of graph database models*. ACM Computing Surveys, 40(1), pp.4-6.

[15] Wikipedia 2020. Graph Traversal. [online] Available at: https://en.wikipedia.org/wiki/Graph_traversal [Accessed 18 July 2020].

[16] Johnston, W., Hanna, J. and Millar, R., 2004. Advances in dataflow programming languages. ACM Computing Surveys, 36(1), pp.1-34.

[17] Rogers, G., n.d. Visual programming with objects and relations. [Proceedings] 1988 IEEE Workshop on Visual Languages,.

[18] MIT Media Lab. *Scratch*. https://scratch.mit.edu/about/

[19] Microsoft Research. *Kodu*. https://www.kodugamelab.com/about/

[20] Epic Games. *Unreal Engine*. https://www.unrealengine.com/en-US/

[21] IBM Emerging Technology. *NODE-RED*. https://nodered.org/about/

[22] En.wikipedia.org. 2020. Active Directory. [online] Available at: [Accessed 14 December 2020].

[23] Amazon Web Services, Inc. 2020. Amazon Web Services (AWS) - Cloud Computing Services. [online] Available at: [Accessed 14 December 2020].

[24] Azure.microsoft.com. 2020. Molnbaserade Databehandlingstjänster | Microsoft Azure. [online] Available at: [Accessed 14 December 2020].

[25] TinkerPop, A., 2020. Apache Tinkerpop. [online] Tinkerpop.apache.org. Available at: [Accessed 14 December 2020].

[26] Blender Foundation. *Blender*. https://www.blender.org/about/

[27] Adobe.com. 2020. UI/UX Design And Collaboration Tool | Adobe XD. [online] Available at: [Accessed 20 August 2020].

[28] Reactjs.org. 2020. React – A Javascript Library For Building User Interfaces. [online] Available at: https://reactjs.org/ [Accessed 26 June 2020].

[29] Martin, B. *Universal Methods of Design* Beverly, United States. Rockport Publishers. 2012.

[30] Elastic NV. *Elastic Kibana*. https://www.elastic.co/kibana

[31] Elastic NV. *Elastic Search*. https://www.elastic.co/elasticsearch/

[32] Spencer, D., 2009. Card Sorting. Brooklyn, New York: Rosenfeld Media.

[33] Nielsen, J., 2020. Mental Models And User Experience Design. [online] Nielsen Norman Group. Available at [Accessed 3 September 2020].

[34] Microsoft.com. 2020. Chatt, Möten, Samtal, Samarbete | Microsoft Teams. [online] Available at: <https://www.microsoft.com/sv-se/microsoft-365/microsoft-teams/group-chat-software> [Accessed 13 December 2020].

[35] Who.int. 2020. Coronavirus Disease (COVID-19) – World Health Organization. [online] Available at: [Accessed 14 August 2020].

[36] Google Developers. 2020. Blockly | Google Developers. [online] Available at: [Accessed 5 September 2020].

[37] Flowgorithm.org. 2020. Flowgorithm - Flowchart Programming Language. [online] Available at: <http://www.flowgorithm.org/> [Accessed 10 October 2020].

[38] Flowhub.io. 2020. Flowhub [online] Available at: [Accessed 5 September 2020].

[39] En.wikipedia.org. 2020. Regular Expression. [online] Available at: [Accessed 14 December 2020].

[40] Carroll, J., 1995. *Scenario-Based Design: Envisioning Work And Technology In System Development*. 1st ed. New York: J. Willey.

[41] Harley, A., 2020. Visibility Of System Status. [online] Nielsen Norman Group. Available at: [Accessed 26 September 2020].

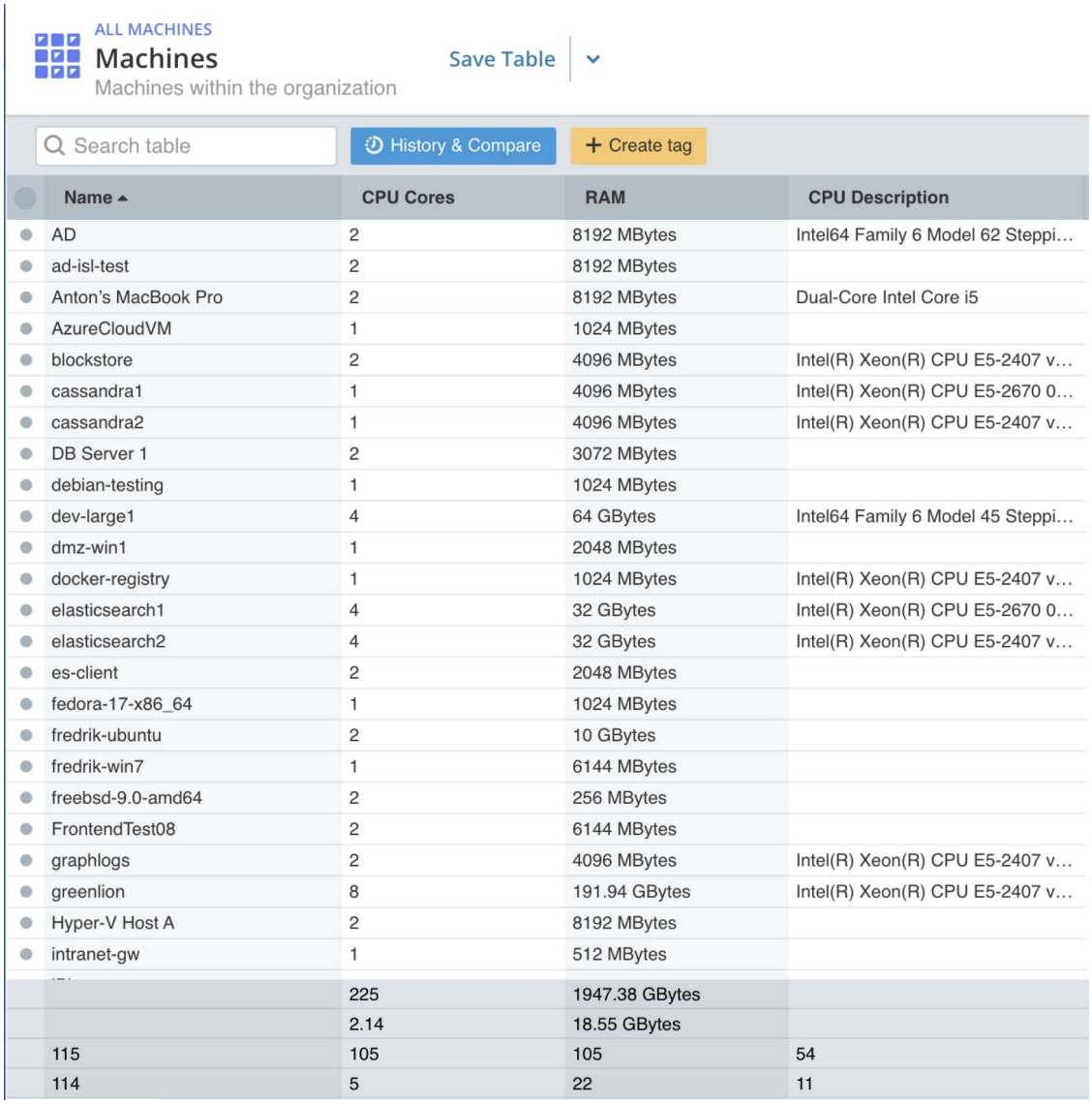
[42] Nielsen, J., 1994. 10 Usability Heuristics For User Interface Design. [online] Nielsen Norman Group. Available at: [Accessed 2 September 2020].

[43] Budiu, R., 2014. Memory Recognition And Recall In User Interfaces. [online] Nielsen Norman Group. Available at: [Accessed 13 September 2020].

[44] Bailey, W., Allan, W., Raiello, P. *Usability Testing vs. Heuristic Evaluation: A Head-to-Head Comparison*. Proceedings of the Human Factors and Ergonomics Society Annual Meeting, 1992.

[45] Nielsen, J., 1994. How to Conduct a Heuristic Evaluation. [online] Nielsen Norman Group. Available at: [Accessed 2 September 2020].

# Appendices




| Name ▲ | CPU Cores | RAM | CPU Description |
|---|---|---|---|
| AD | 2 | 8192 MBytes | Intel64 Family 6 Model 62 Steppi… |
| ad-isl-test | 2 | 8192 MBytes | |
| Anton's MacBook Pro | 2 | 8192 MBytes | Dual-Core Intel Core i5 |
| AzureCloudVM | 1 | 1024 MBytes | |
| blockstore | 2 | 4096 MBytes | Intel(R) Xeon(R) CPU E5-2407 v… |
| cassandra1 | 1 | 4096 MBytes | Intel(R) Xeon(R) CPU E5-2670 0… |
| cassandra2 | 1 | 4096 MBytes | Intel(R) Xeon(R) CPU E5-2407 v… |
| DB Server 1 | 2 | 3072 MBytes | |
| debian-testing | 1 | 1024 MBytes | |
| dev-large1 | 4 | 64 GBytes | Intel64 Family 6 Model 45 Steppi… |
| dmz-win1 | 1 | 2048 MBytes | |
| docker-registry | 1 | 1024 MBytes | Intel(R) Xeon(R) CPU E5-2407 v… |
| elasticsearch1 | 4 | 32 GBytes | Intel(R) Xeon(R) CPU E5-2670 0… |
| elasticsearch2 | 4 | 32 GBytes | Intel(R) Xeon(R) CPU E5-2407 v… |
| es-client | 2 | 2048 MBytes | |
| fedora-17-x86_64 | 1 | 1024 MBytes | |
| fredrik-ubuntu | 2 | 10 GBytes | |
| fredrik-win7 | 1 | 6144 MBytes | |
| freebsd-9.0-amd64 | 2 | 256 MBytes | |
| FrontendTest08 | 2 | 6144 MBytes | |
| graphlogs | 2 | 4096 MBytes | Intel(R) Xeon(R) CPU E5-2407 v… |
| greenlion | 8 | 191.94 GBytes | Intel(R) Xeon(R) CPU E5-2407 v… |
| Hyper-V Host A | 2 | 8192 MBytes | |
| intranet-gw | 1 | 512 MBytes | |
| | 225 | 1947.38 GBytes | |
| | 2.14 | 18.55 GBytes | |
| 115 | 105 | 105 | 54 |
| 114 | 5 | 22 | 11 |

**Figure 1:** Table explorer

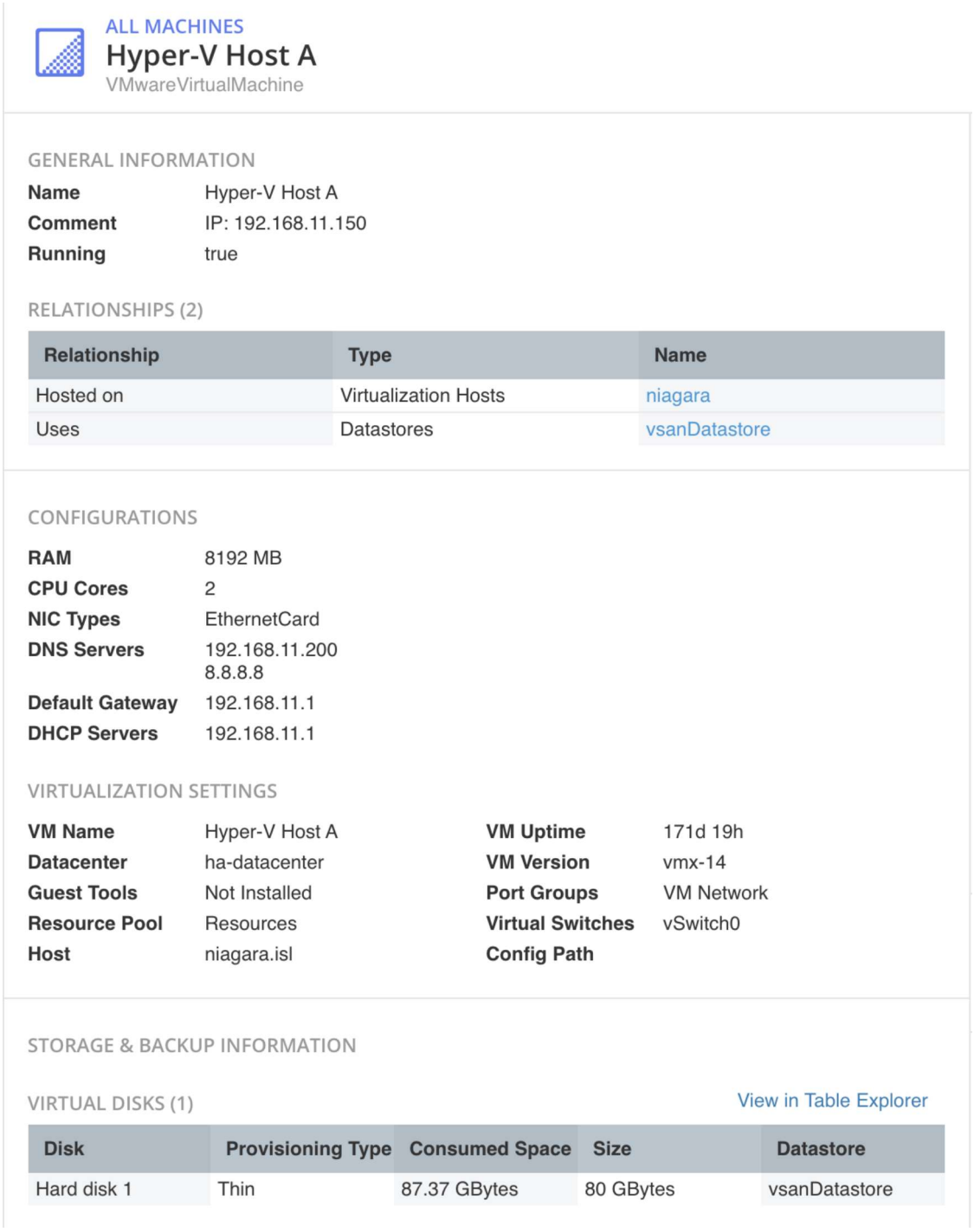



**Figure 2:** Property page

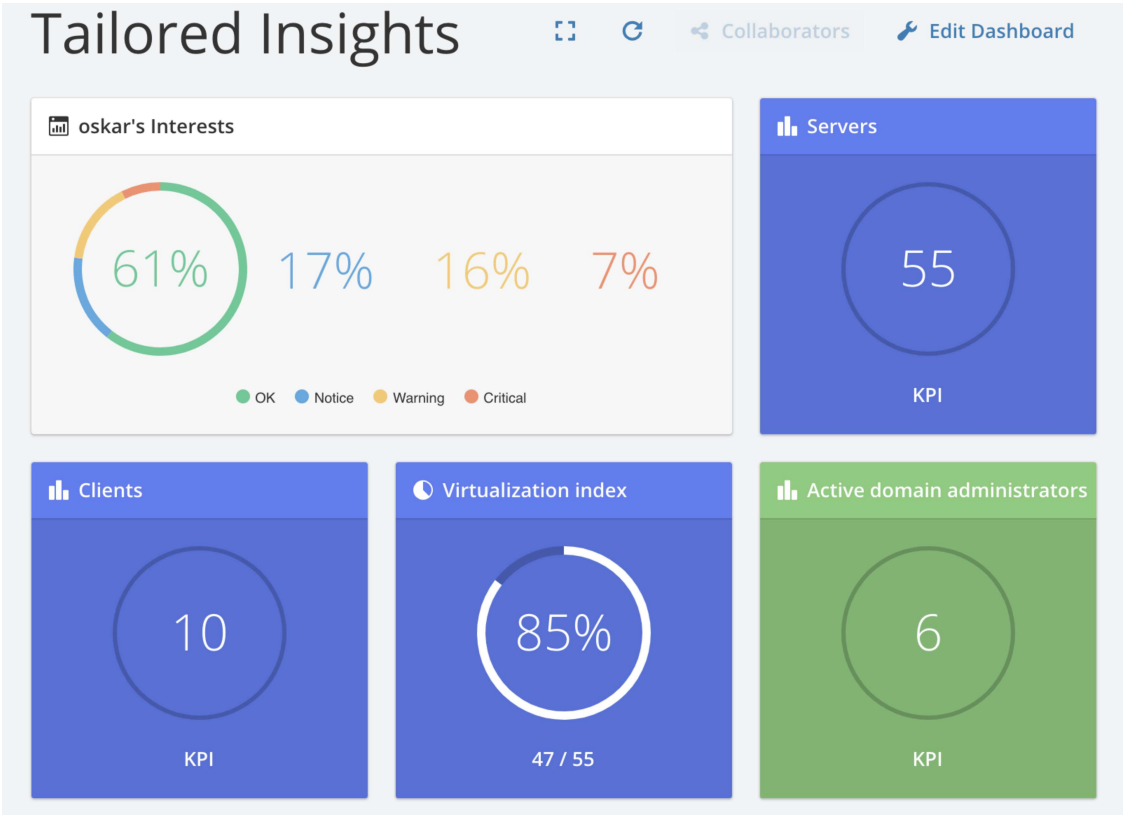



**Figure 3:** Dashboard

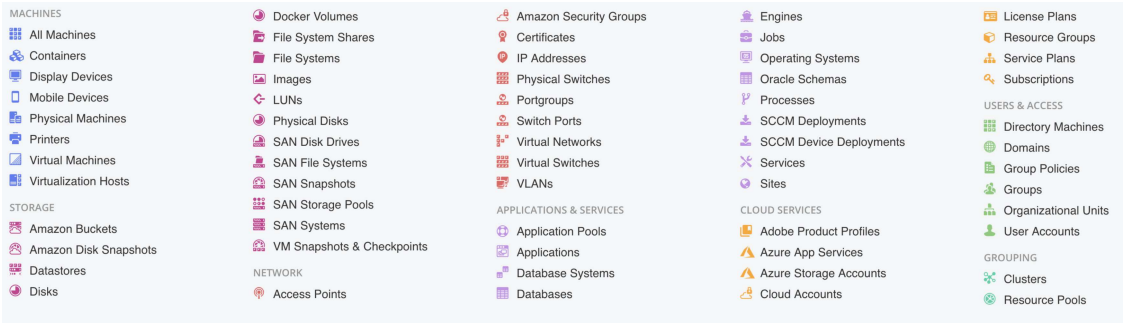



**Figure 4:** Resource types

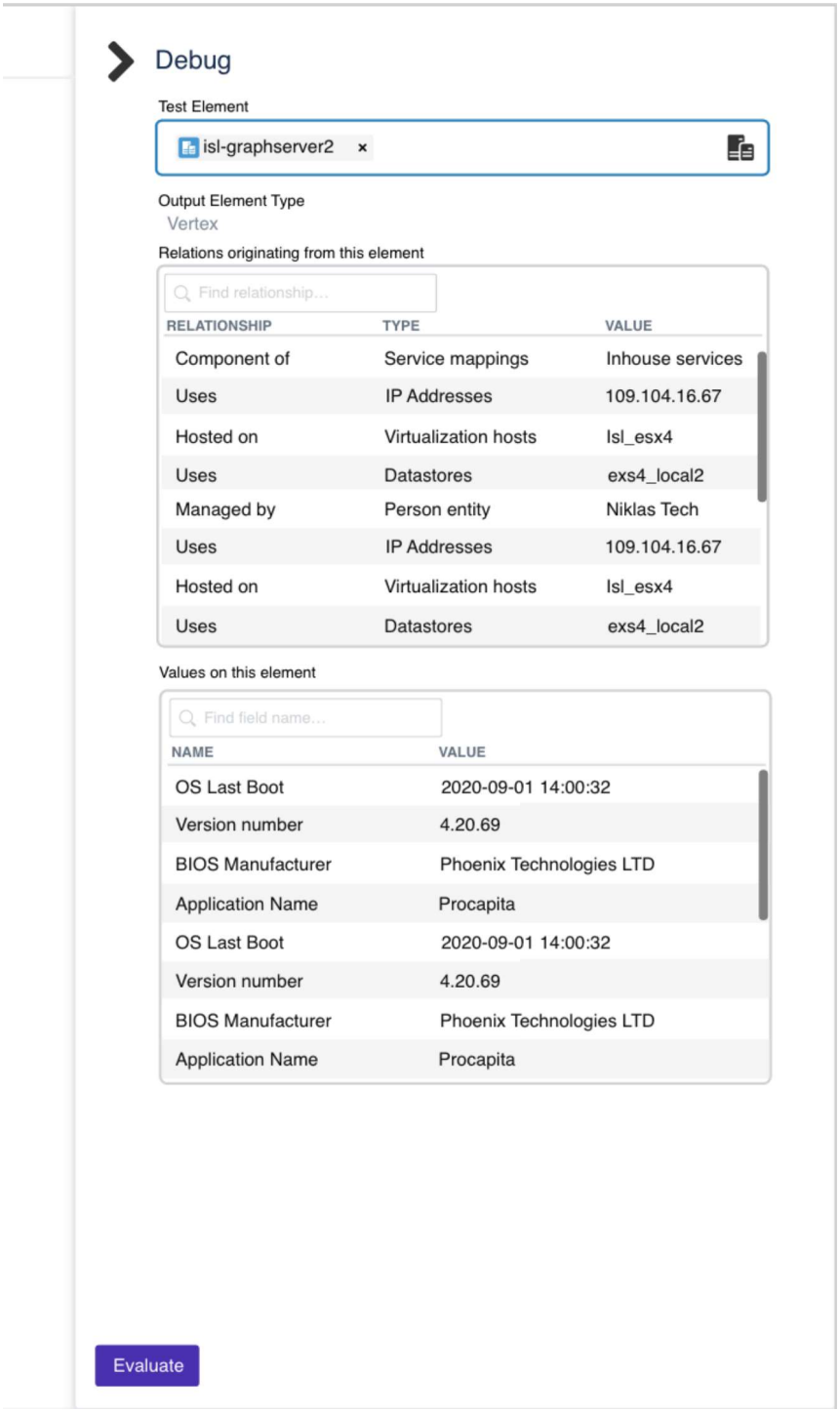



**Figure 5:** Resource types