BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE IN ENGLISH

DIPLOMA THESIS

Adaptive User Interfaces for Decision Making

Supervisor
Lect. Dr. Cojocar Dan

Author
Udrea Horațiu Alexandru

2021

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ ÎN LIMBA ENGLEZĂ

LUCRARE DE LICENŢĂ

Interfețe de Utilizator Adaptabile în Luarea Deciziilor

Conducător științific
Lect. Dr. Cojocar Dan

Absolvent
Udrea Horațiu Alexandru

2021

ABSTRACT

Drawing inferences from large amounts of data with a fixed structure can be efficiently done by computers when the purpose is clear and the algorithm is known. When these preconditions are not met, however, people remain the only available option, though they usually lack the ability to remember or comprehend all the data in a timely manner. In this case, tools and methods that facilitate the short-term retention and comprehension of data accelerate the cognitive inference process and help with solving this problem.

The aim of this thesis is to describe an original method of creating such a data-driven tool, an adaptive user interface that facilitates human decision-making based on data with a fixed structure. This underlying data is dynamically clustered in a hierarchical manner and the user has complete control over the way this hierarchy is created and how the interface adapts to it.

We begin by introducing the reader in the context of the problem and continue by presenting topics and relevant work from various related fields of study. Afterwards, we describe the method for solving the problem in detail, giving appropriate examples, explaining the reasoning behind the procedures and describing a possible implementation of the system. Finally, we present the conclusions of the thesis and explore ideas for future work on the subject.

Contents

Chapter 1

Introduction

In order to draw inferences from large amounts of data with a fixed structure such as tables with lots of rows, people usually are at a disadvantage compared to computers because of their inability to remember or comprehend all the data in a timely manner. However, when the purpose is not clear enough or the algorithm is not yet implemented, the intelligence of the person that needs to draw the conclusions becomes the only viable option. In this particular case, any method of facilitating the comprehension and short-term retention of data speeds up the cognitive inference process and helps with solving the problem.

Examples of this kind of problem include:

- Checking resource allocations to projects and deciding upon necessary changes in order to accomplish different goals such as finishing the projects in time or grouping certain resources together.
- Finding a weekday for going on a trip, having a fixed schedule given as a set of activities and therefore taking into account the missed activities, their importance and the possible impact on the activities in the next day.
- Checking the locations of objects in a warehouse and deciding upon necessary rearrangements, taking into account the objects' type, weight and size.

The aim of this thesis is to provide an original data-driven solution for the impediment created by large amounts of data items with fixed structure in the context of decision-making. The method implies creating software with an adaptive user interface where the underlying data is dynamically clustered in a hierarchical manner. The user has complete control over the generation of this hierarchy and the way in which the interface adapts to it. This is done in order to allow anyone to adjust the software and suit it to his own way of reasoning about the problem, therefore accelerating the process of deduction and improving the required time for finding a solution.

The remaining content of the thesis is structured as follows:

- **Chapter 2** presents topics and related work from various fields of study relevant to the presented problem and the proposed solution.
- **Chapter 3** presents the method for solving the problem in detail, giving relevant examples and explaining the reasoning behind the procedures. It then continues by describing the implementation of a system with an adaptive user interface that facilitates decision-making based on datasets in the form of tabular data.
- **Chapter 4** presents the conclusions of this study, outlining the advantages and limitations of the method and exploring ideas for future work on the subject.

Chapter 2

User Interfaces and Data-Driven Inference

This chapter presents topics and related work from various fields of study relevant for the presented problem and for the solution described in Chapter 3.

2.1 User Interfaces

User Interfaces [Ste00] are the means by which humans interact with computers. The goal of these interfaces is to allow the operator to control the machine while receiving feedback for the commands given and ongoing processes. This feedback facilitates further interaction with the computer and gives the user the necessary information for making decisions regarding the issue that needed to be solved using the device. Generally, it is desirable that user interfaces make these interactions easy, efficient and enjoyable (user-friendly), maximizing usability. Therefore, designing such components usually involves knowledge from related domains such as ergonomics and psychology.

In the history of human-computer interaction [DFAB04], user interfaces took various forms, from batch interfaces (punched card readers and line printers) to command-line interfaces (keyboard and monitor, real-time interaction) and graphical user interfaces (GUIs), capable of displaying graphics. The most common user interface today includes a GUI, audio input and output devices, and a set of tactile input devices, usually a touch-based input screen or mouse and keyboard setup.

User interfaces have evolved to a great extent during recent years, along with the hardware computing power and size. This evolution led to increased accessibility and availability of these kinds of computing devices to the point where non-technical people can utilize them to facilitate daily life tasks. Home desktop computers are already widespread and smartphones are capable of rendering high-quality graphics, record and playback audio, and use a touchscreen for input-output while having a small enough size to fit in a pocket. Because the physical interface is so versatile, developers of user interfaces only need to create software used on the end user's device to control the physical components and manage the human-computer interaction. Since the product has to suit the needs of numerous different people that lack the knowledge and technical skill to make necessary changes, it is commonly required that the interfaces are easily configurable by the user and able to suit different special needs (eyesight issues, hearing issues, etc.), which most of the times proves challenging and time-consuming.

2.2 User Experience Design

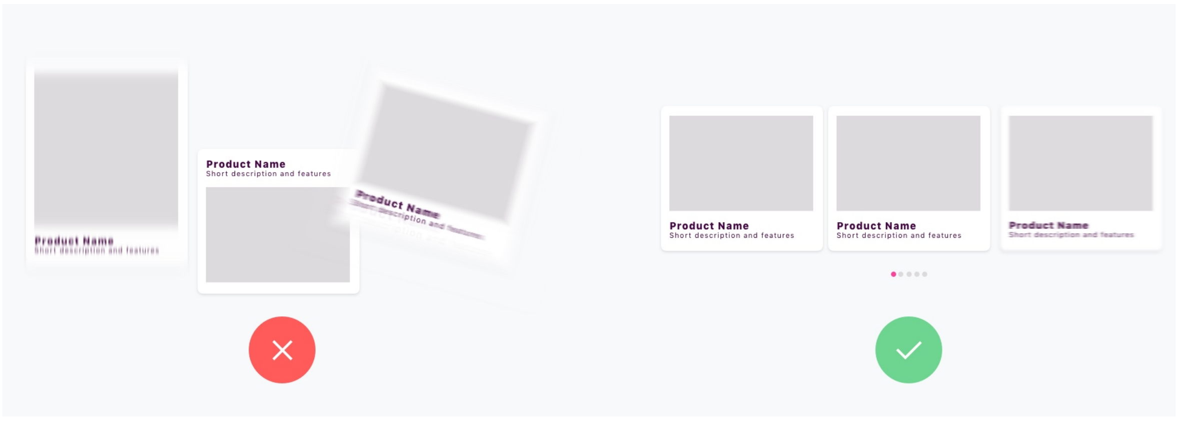

Product Name
Short description and features
Product Name
Short description and features
Product Name
Short description and features
Product Name
Short description and features
Product Name
Short description and features
Product Name
Short description and features

Figure 2.1: Common Fate design principle [Prz19]

Spatial memory is an important facet of human cognition – it allows users to learn the locations of items over time and retrieve them with little effort [SCG13]. In human-computer interfaces, knowledge of the spatial location of controls can enable a user to interact fluidly and efficiently with the interface without needing to perform a slow visual search. Computer interfaces should therefore be designed to provide support for developing the user's spatial memory, allowing the user to exploit them for rapid interaction whenever possible. However, existing systems offer varying support for spatial memory. Many break the user's ability to remember spatial locations, by moving or re-arranging items (as in Figure 2.1), whereas others leave spatial memory underutilized, requiring slow sequences of mechanical actions to select items rather than exploiting the user's strong ability to index items and controls by their on-screen locations.

Evidence that spatial memory is a particularly powerful capability of the human brain can be found in mnemonic literature and dates back thousands of years.

The ancient Greeks and Romans used spatial mental organizations based on the architecture of the time, known as *memory palaces* [Figure 2.2], to connect, organize, and memorize unfamiliar ideas, particularly for public speaking. This was called *the method of loci* [Hed17]. By embedding key images representing topics in a mental representation of a spatial environment, such as the rooms in a familiar building, orators were able to memorize extremely long sequences of topics. These could then be retrieved by mentally walking through the building and viewing the images in their respective spatial locations.

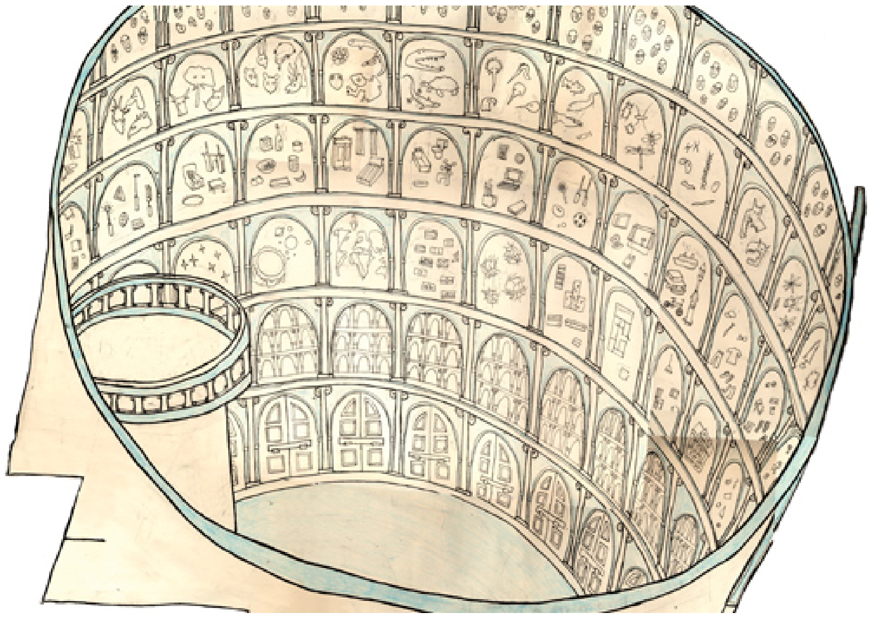


Figure 2.2: Illustration of the memory palace technique [Hed17]

In human-computer interaction, spatial memory provides many of the same benefits as in the real world: strong spatial knowledge of interface layouts and control locations, especially in graphical user interfaces, allows users to substantially reduce the cognitive and physical effort required for interaction. Evidence for the benefits provided by spatial memory can be found in the strong correlation between measures of spatial ability and interface performance. Users who are unfamiliar with an interface must spend considerable time searching for controls because the time to perform a visual search is proportional to the number of items. In contrast, users who are familiar with a spatially stable interface do not need to carry out the visual search, and can instead simply retrieve item locations. This is much faster than searching because retrieval time is a logarithmic function of the number of items.

Furthermore, extensive spatial knowledge of an application's controls enables interaction automaticity, which substantially frees the user's cognitive resources from the need to consider interface mechanisms, allowing the user to instead focus on higher-level task considerations. Spatial knowledge of the locations of controls can also decrease the frustration that arises from the need to search for unfamiliar controls or controls that have moved.

As useful guidelines for designing user experience, one can follow the Gestalt Principles [Prz19] where the key fundamental rules are emergence, reification, multistability, and invariance. The principles are:

- **Proximity principle** - We perceive that elements that are close to each other appear to be related
- **Similarity Principle** - Elements that have similar visual appearance seem to be more related or grouped than the ones not sharing the same attributes.
- **Continuation Principle** - Our perception tends to see objects arranged in lines or curves as more related or grouped.
- **Closure Principle** - Objects are often perceived as a whole thing, even when they are incomplete.
- **Symmetry Principle** - Our mind perceives symmetrical objects as parts of the same group.
- **Common Fate Principle** [Figure 2.1] - When the elements tend to move in the same direction we perceive them as part of the same group.
- **Figure-Ground Principle** - We know which elements are placed in the foreground and with ones are in the background intuitively.
- **Common Region Principle** - Objects placed within the same region are perceived to be in the same group.
- **Periodicity Principle** [Figure 2.3]- Elements that appear multiple times in similar distances are perceived as related.

2.3 Adaptive User Interfaces

In Human-Computer-Interaction [DFAB04], adaptation is modeled as two complementary system properties: adaptability and adaptivity. Adaptability is the capacity of the system to allow users to customize it from a predefined set of parameters. Adaptivity is the capacity of the system to perform adaptation automatically without deliberate action on the user's part. Whether adaptation is performed on human requests or automatically, the design space for adaptation includes three orthogonal axes [Figure 2.4]: target, means, and time [TC99].

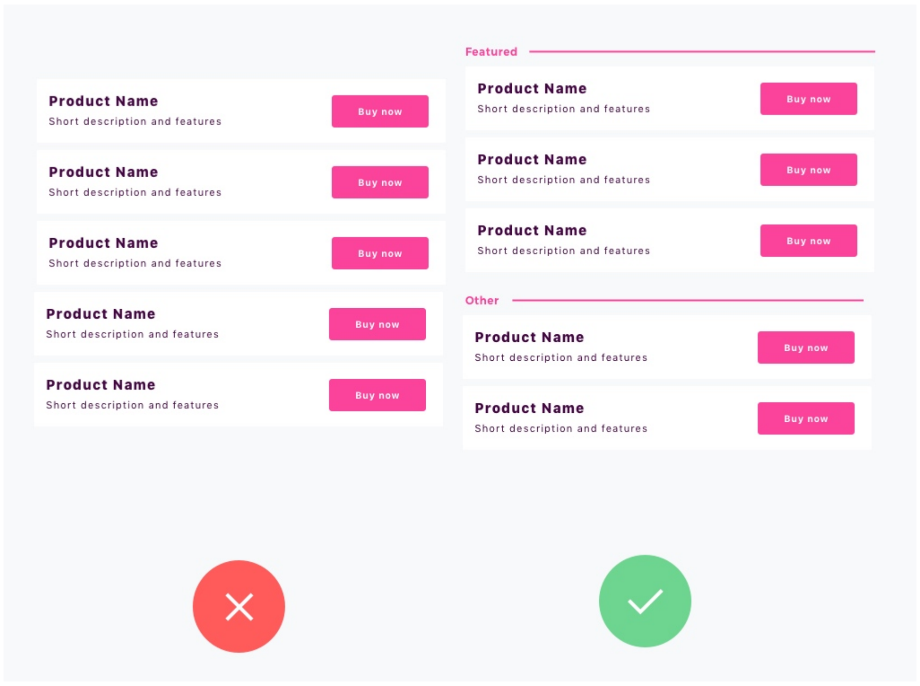


Figure 2.3: Periodicity design principle [Prz19]

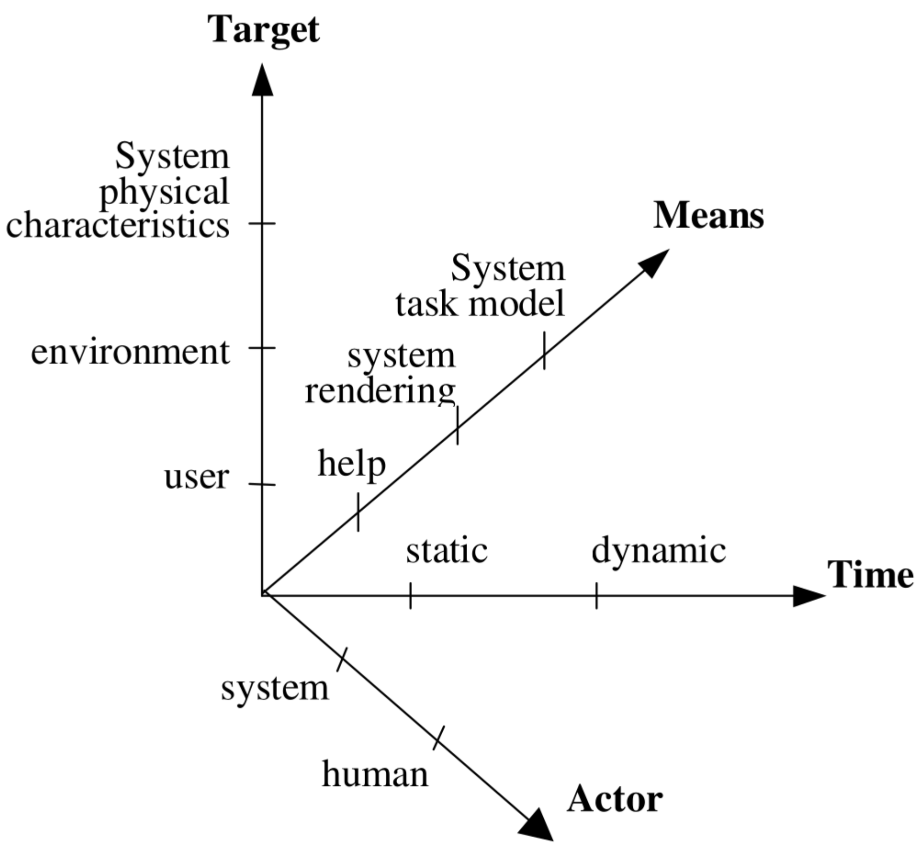


Figure 2.4: A design space for adaptation at a high level of reasoning [TC99]

- **The target for adaptation** - This axis denotes the entities for which adaptation is intended: adaptation to users, to the environment, and to the physical characteristics of the system (e.g. peripheral devices such as a mouse or touch-screen).
- **The means of adaptation** - This axis denotes the software components of the system involved in adaptation: typically, the system task model, the rendering techniques, and the help subsystems may be modified to adapt to the targeted entities. The system task model is the system implementation of the user task model specified by experts on human factors. The rendering techniques denote the observable presentation and behavior of the system. The help subsystems include help about the system and help about the task domain.
- **The temporal dimension of adaptation** - Adaptation may be static (i.e. effective between sessions) and/or dynamic (i.e. occurring at run time).

In recent years there has been significant progress in automated assistance for developers creating user interfaces. Commercially available interface builders, user interface management systems, and interface toolkits provide considerable savings to developers in time and in effort needed to produce a new interface. Even with these commercial tools present, the amount of effort and low-level detail involved in constructing these interfaces remains substantial. Therefore, researchers are investigating techniques to automate more portions of the model-based user interface development. One promising area is that of interface design process [PEGM94].

In this approach, developers work with high-level specifications (models) of the interface to define dialog and layout characteristics. Model-based systems facilitate the automation of interface design tasks. A successful approach has been to use the application's data model to generate the static layout of an interface. An intelligent design tool examines the data model and applies a set of design rules to produce a static design of an interface. Because the data model is shared between the interface design and the target application design, both designs can be coupled and changes to the application design can be propagated easily to the interface design.

In the simplest solution, there are four basic components within the architecture: user's client, interface agent, ontology and knowledge base which stores the data that the user is interested in. The data is stored and organized in a dynamical hierarchy within the ontology and the client provides the user with a means to access it via the software agent technology. To deliver the dynamic user interface, the user chooses the parameters for its creation. Subsequently, the interface agent that possesses communication capability will deliver a dynamic user interface to the client, based on the information stored in the ontology and the user's preferences. The dynamic capabilities of the interface agent help with presenting the information to the users in the desired manner [LL07].

The interface agent interacts with the user by asking him to select a filter on the user interface. Based on the input, the interface agent will generate the next category of desired information from the ontology. This type of interaction will continue until the agent has gathered sufficient information to display the final groups of data.

Traditionally, the term ”ontology” is defined as the study or the science of being. In other words, the agent is able to manipulate the semantic of other knowledge since knowledge is represented by the same vocabulary based on common conceptualization. An ontology enables computer programs and software agents to understand the semantics, thus making it possible for them to process the content. Although different organizations may have their own ontologies, such differences can be overcome by mapping the particular terminology to a shared ontology or by defining direct mappings between ontologies.

2.4 Data Visualization

The process of data visualization includes four basic stages, combined in several feedback loops [War04]. These stages are illustrated in Figure 2.5 and consist of:

- The collection and storage of data itself
- The preprocessing designed to transform the data into something we can understand
- The display hardware and the graphics algorithms that produce an image on the screen
- The human perceptual and cognitive system (the perceiver)

Since the user does not interact with the data directly, his perception might be different from the original data. The goal will always be to make the user perceive the information with a minimal error by optimizing the processes related to these stages.

Data can come in various forms and sizes, and so database systems have grown more and more powerful, being optimized for storing different types of data. However, the power to use the stored information is not easily accessible. Often the interface is very restrictive, using predefined forms, or the database logic has to be accessed by entering queries in the database query language, most often SQL [EMK^{+}04]. In this case, not only the contained data has to be visualized but also

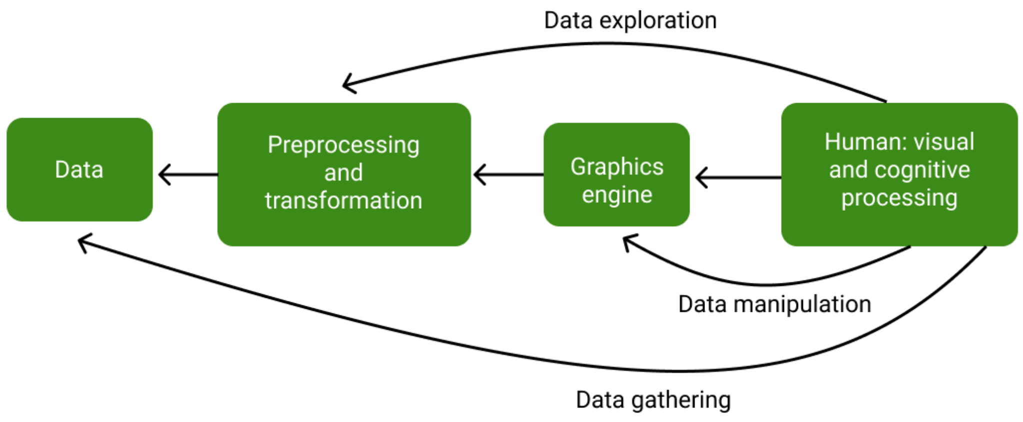

Data exploration
Data
Preprocessing and transformation
Graphics engine
Human: visual and cognitive processing
Data manipulation
Data gathering

Figure 2.5: A schematic diagram of the visualization process based on [War04]

the information associated with its retrieval and modification. Given the proximity of the logic of relational databases and formal logic, a graphical logic system similar to the ones being developed in the context of Conceptual Graphs should be both easy and powerful enough to support users in their interaction with databases. Nested Concept Graphs [DC03] are able to solve this issue by providing visual concept bindings for SQL.

The basic idea of the Nested Concept Graphs is to provide a universal database query language that is more accessible to users than SQL. In this sense, it may be considered similar to the visual query languages available in many of the commercial database management systems. However, looking more closely, those interfaces provide only a graphical representation of a subset of SQL, while letting the more complicated features still be entered in a textual way. It is argued that the consistent presentation of queries in form of the Nested Concept Graphs leads to a more intuitive interface.

A very simple example might be the following statement:

```
SELECT Lastname FROM Person WHERE Firstname='Grit'
```

which means that the user wants to get the last names from the table Person, where the first name is 'Grit'. The translation into Concept Graphs looks like the one in Figure 2.6.

This graph is as easy to understand to everyone used to reading Conceptual Graphs and does not require expertise in the database domain, therefore making the visualization of data retrieval and manipulation operations more accessible.

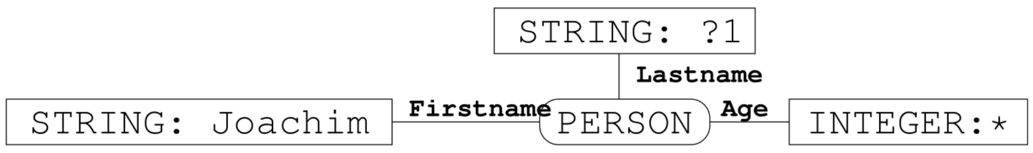


Figure 2.6: A simple query as Conceptual Graph [DC03]

2.5 Information Chunking

Chunking [TSO19] is a method to visualize information by clustering smaller units of information into larger units that are easier to remember [Figure 2.7]. It is argued that it helps exceed the limited capacity of working memory in the human brain, therefore improving short-term retention.

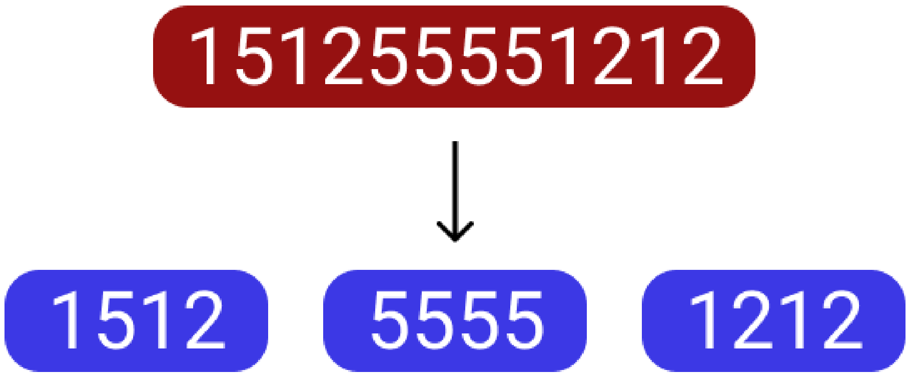


Figure 2.7: Sequence of digits split into chunks that are easier to remember

Across four experiments described in *"How does chunking help working memory?"* [TSO19], chunking was found not only to benefit recall of the chunked information but also of other non-chunked information held at the same time in the working memory, supporting the assumption that chunking reduces the load.

The chunking benefit was found to be independent of the chunk size only if the chunks were composed of unique elements, such that each one of them could be represented by its first element, but not in the case where several chunks consisted of overlapping sets of elements. This effect is not caused by differences in practice duration as it persisted even when the participants were required to speak while being presented with the items to remember, therefore inhibiting memory performance (a process known as *articulatory suppression*). Hence, working memory capacity is not limited to a fixed number of chunks regardless of their size.

Furthermore, chunks that were remembered first improved recall of other, not-chunked material, but chunks that were remembered last did not. It was concluded that a chunk reduces the load on the working memory by using a compact representation of the chunk version from long-term memory, which replaces the individual elements normally stored in the working memory, therefore freeing up capacity for subsequently encoded material.

Visual working memory capacity is also influenced by the expertise effect [SWD+20]. This capacity is usually defined as the number of memory representations on which an individual can focus at a given time. Its limits and the conditions under which they can be exceeded have been comprehensively studied. The upper limit of this capacity is generally accepted as being at most four representations. However, it has been debated whether this capacity is best defined as a limit on independent representations or as more complex linked structures. The reported capacity estimates seem low and would suggest that both experts and novices should struggle with maintaining accurate representations of complex displays in memory. But the granularity of a memory representation can vary greatly with the comprehension of a given representation and therefore varies with expertise. This relationship apparently contributes to the seemingly large working memory capacity of experts: While experts, like novices, can maintain approximately the same number of representations in working memory, expert representations are significantly more complex when they involve domain knowledge. Consequently, experts can focus on a larger amount of information than novices, allowing superior response time, information recall and problem-solving.

Research on the nature of expertise in a knowledge domain has demonstrated that experts can encode and recall large amounts of information in a complex visual display as long as the display is relevant to their expertise. When viewing domain-relevant stimuli, experts do not encode individual components of the stimuli as separate memory representations. Rather, they encode the relationships between the components as a single representation through information chunking. This feat is illustrated with an example from chess. The layout of the chess pieces in Figure 2.8 corresponds to a *bishop and knight mate* checkmate pattern. While a chess novice might encode the image as four separate representations that contain the identity and location features for each chess piece on the board, the expert encodes a single informational chunk (*bishop-and-knight-mate*) that contains the identity, relative location features and additional cues, such as how to avoid or achieve the pattern and the consequence on the game outcome. The maintenance of the single chunk as opposed to the four piece-location pairs reduces the load on working memory and frees additional resources for the expert, which in turn improves task performance.

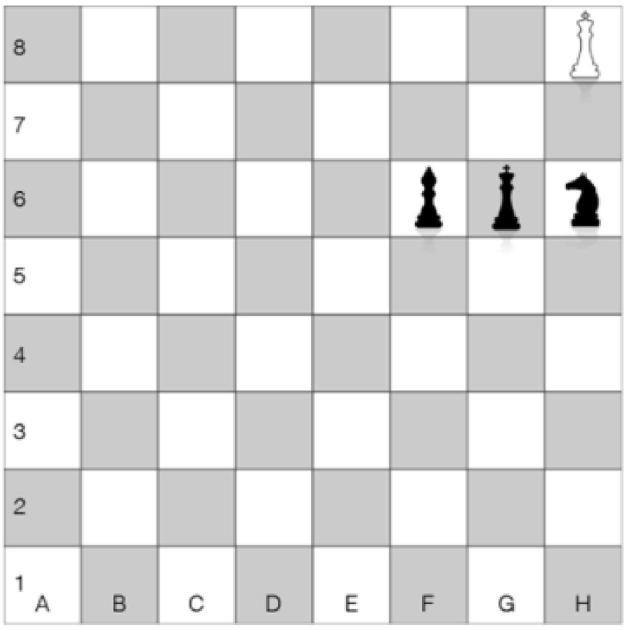


Figure 2.8: Chessboard configuration of *bishop and knight mate* checkmate pattern [SWD$^+$20]

In a similar manner, the retention of symbolic circuit drawings was explored using skilled electronics technicians and novice subjects across three experiments in *Chunking in recall of symbolic drawings* [ES79]. In the first experiment, a skilled technician reconstructed circuit drawings from memory, using chunking by functional units similar to chess master's recall of chess positions. In the second experiment, skilled technicians were able to recall more than novice subjects following brief exposure to the drawings. They did not have this advantage for randomly arranged symbols. In the third experiment, the size of the chunks retrieved systematically increased with additional study time. In-depth analyses suggested that the skilled subjects identified the conceptual category for an entire drawing and retrieved elements using a generate-and-test process.

2.6 Data-Driven Inference

A Data-Driven [Str12] process is, as the name suggests, a process driven by data. Data, usually in large amounts, is used to obtain insights and ultimately make decisions that help achieve the established purpose. This process can be performed either by a human or by a computer that makes use of artificial intelligence and statistical methods. Even though in this information age data has become abundant and data-driven approaches seem to be new and revolutionary, upon closer study these methods are similar to the ones used in the past for science and research purposes.

Renaissance naturalists were no less inundated with new information than our contemporaries. The expansion of travel exposed European naturalists to new facts that did not fit into the systems of knowledge inherited from the Greeks and Romans. This prompted those interested in understanding the natural world to devise new methods for managing this data, such as note-taking and new classification systems. Ironically, these methods and systems, which were meant to tame the information overload, made it possible to accumulate even more data. But accumulation was usually only a means to an end. These early naturalists established collections, which included specimens, drawings and texts so that they could compare these items systematically and draw conclusions about the natural world from the comparisons. In general, they were not testing specific hypotheses, but trying to bring order to the perplexing diversity of natural forms by examining large amounts of collected ”data”. This tradition continues to be central in natural history to the present day. Consequently, the natural historical sciences have been fundamentally data-driven all along.

All these sciences, from early modern natural history to contemporary post-genomics, have essentially been comparative, identifying similarities, differences, patterns, and clusters. A recurring concern among them, past and present, has been the production and enforcement of standards. Because comparative approaches are so crucial to data-driven sciences, the uniformity of the data has been essential. For data to serve as a basis for grouping or dividing samples into different classes, they need to refer to the same property in all samples. These properties of the data set can either be deduced from the context from which the data is extracted or inferred statistically.

For human data-driven inference, the most notable example is *data-driven learning* [Joh91], especially foreign language learning, where the learner is given access to real, correlated linguistic examples. If we take this approach we do not attempt to make the system intelligent: we simply provide the evidence needed to answer the learner’s questions and rely on the learner’s intelligence to find answers. The most important computing tool for this data-driven approach is the concordancer, which is able to recover from text all the contexts for a particular item (morpheme, word, or phrase) and to print them out in a way that facilitates rapid scanning and comparison. The most usual format is the keyword-in-context (KWIC) concordance in which the keywords are arranged one below the other, down the center of the page, with a fixed number of characters of context to the left and right. A useful refinement, particularly where the result involves regularities and patterns in large numbers of citations, is the ability to sort alphabetically the contexts to the left or right of the keyword so that similar contexts are grouped. What is novel about this method is the perception that ”research is too serious to be left to the researchers”:

that the language-learner is also, essentially, a research worker whose learning needs to be driven by access to linguistic data - hence the term *data-driven learning* (DDL) to describe the approach.

The use of the concordancer can have a considerable influence on the process of language learning, stimulating inquiry and speculation on the part of the learner and also helping him develop the ability to see patterns in the target language and to form generalizations that account for those patterns. These benefits are often claimed for other inductive approaches to language learning that require the student to move from data to generalization. Traditionally, these have been based on ″rule-hiding″: the material's writer decides what rule or rules are to be taught and writes a set of examples (sentences or pseudo-text) to encapsulate them. The task of the learner is to work in the opposite direction and to recover the rules from the examples. It is, perhaps, not surprising that despite its supposed advantages, language-teaching based on rule-hiding may show no discernible advantage over the more traditional deductive approach of ″rule first, practice second″. What is distinctive about the DDL approach to inductive language teaching is the principle that the data is primary, and the teacher does not know in advance exactly what rules or patterns the learners will discover: indeed, they will often notice things that are unknown not only to the teacher but also to the standard works of reference on the language. It is this element of challenge and discovery that gives DDL its special flavor and stimulus.

2.7 Enterprise Resource Planning Systems

Enterprise Resource Planning (ERP) Systems [O'L00] make heavy use of data processing and visualization to improve business efficiency. Besides data management, a key aspect of the user interfaces in those systems is the ability to facilitate quick decision-making based on the presented information.

The usability of ERP systems can influence end users' attitude and behavior to use IT. A survey was conducted (presented in *″Does usability matter? An analysis of the impact of usability on technology acceptance in ERP settings″* [SMR16]) with ERP users from an organization in the heavy metal industry in Bangladesh. The survey findings empirically confirmed that interface usability has a significant impact on users' perceptions of usefulness and ease of use which ultimately affects attitudes and intention to use the ERP software.

ERP systems have long been criticized regarding their complexity. This complexity is due to the integration of different business applications and the processing of huge amounts of data. Furthermore, the functionality and complexity of systems create confusion and frustration, which results in negative responses from users and mistakes made while working with such applications.

A possible solution to improving the usability of a software system lies in the ability to personalize the UI. One such approach which addresses this problem is adaptive user interfaces (AUIs) [SW09]. AUIs are user interfaces that are designed to personalize and improve user interaction. Such an interface aims to improve the exchange of information between the user and the computer in a way that meets the user's requirements, goals and context of use. They are capable of meeting these requirements by presenting personalized views to match the needs of an individual user. This is achieved by reducing the complexity of the UI and making it easy to learn and use whilst improving overall user satisfaction.

Mobile access to information stored in corporate back-end systems such as ERP systems is particularly beneficial for mobile workers. They can interoperate with such a system from anywhere and at any time. However, the preparation of content targeted to various mobile devices is a complex task. It requires special adaptation methods or at least specific design principles [KJN06]. Designing and developing the user interface is a complex task because of the characteristics of mobile devices and the mobile environment. The structure of the user interface can be adapted by splitting available information into smaller fragments that are displayed in separate groups and ordered based on chosen criteria. A vision is that future users will be able to access traditional desktop-based office applications from their mobile phones in a completely transparent way, with a complex automated adaptation process running behind the scenes.

Chapter 3

Creating Adaptive User Interfaces

This chapter first presents a method for solving the problem by describing its steps, rules and algorithms, giving appropriate examples and explaining the reasoning behind the procedures. It then continues by describing the implementation of a system that uses the proposed method to create adaptive user interfaces that facilitate decision-making based on a set of data items with a fixed structure.

3.1 Proposed Method

3.1.1 Main Idea

For helping the user make decisions, we follow a purely data-driven approach similar to the one described in *"Should you be persuaded: Two samples of data-driven learning materials"* [Joh91], where we do not attempt to make the system intelligent: we only provide the data in an organized manner, relying on the user's intelligence to make the necessary deductions. We achieve this in four steps:

- **Defining the relationships** between data properties, described in Section 3.1.2
- **Data Modeling** - transforming the tabular data into a tree structure, described in Section 3.1.3
- **Data selection and sorting** - choosing what data needs to be displayed and in which manner, described in Section 3.1.4
- **Displaying the data** - adapting the user interface to the organized data, described in Section 3.1.5

During this chapter, we use the example dataset defined in Table 3.1, representing items from a warehouse that have to be shipped next week. Each item is represented by an associated unique identifier, the type of the item, the day on which it has to be delivered, its location in the warehouse (as an area code) and the employee responsible for it. This data might be the result of a series of table joins from a relational database, which is quite common.

Id	Item type	Delivery day	Location	Responsible employee
1	Chair	Tuesday	L1	George
2	Chair	Monday	L10	Ryan
3	Bed	Friday	L20	Ryan
4	Chair	Tuesday	L10	George
5	Bed	Monday	L10	Ryan
6	Chair	Tuesday	L10	George
7	Bed	Friday	L20	George

Table 3.1: Warehouse example dataset

We describe two types of decisions that need to be taken from this dataset and use them as the goals for the upcoming subsections:

- **Goal 1** - Each item type has its own equipment necessary for transporting it from its location to the delivery point in the warehouse. This equipment can carry more items of the same type. We would like to check, on each delivery day, for each item type, the locations in which the items of that type reside. This can help the allocation of equipment for those locations and also checking that the items are properly clustered. Depending on the capacity of the equipment, we might decide to make some improvements before the delivery date, like moving some items to other locations to avoid using the equipment for just a few objects when the capacity is larger, thus minimizing the time needed for delivery.

- **Goal 2** - The employee responsible for an item has to be at its location when it is picked up for delivery in order to check that it is handled accordingly. We want to see for each employee, in each delivery day, how the items that they are responsible for are distributed across the warehouse's locations. We can check how much they have to move around and take actions to improve the process by considering other metrics, such as how fast the person usually goes or if they have a motion impediment.

3.1.2 Defining Relationships

In our case, ontology modeling makes use of relationships between the properties of the dataset (also referred to as a knowledge base in this case [LL07]). In order to facilitate grouping in the next steps, we define nesting relationships between the dataset's properties (not between the data itself), usually the columns of the original table.

An example of a composition relationship is between the properties *year*, *month* and *day*: a year consists of multiple months and a month is composed of multiple days. So naturally, you would define the nesting relationship as: *for each year there are some months, and for each of those months from that specific year, there are some days*. We shall denote this relationship as $year \rightarrow month \rightarrow day$. This nesting relationship is a good example because it seems natural, being defined by the creators of this time measurement system. However, to facilitate flexibility, the ontology should be able to account for every user-defined nesting relationship. This means that $month \rightarrow year \rightarrow day$, even though counter-intuitive, should be a valid nesting relationship. In plain english, you could translate it as: *for each month there are some years, and for each of those years linked to that specific month, there are some days*. Although it seems unnatural, people that seek certain knowledge would benefit from this type of organization. This becomes clearer when we create nesting relationships for our two example goals, which do not have a "natural" nesting relationship.

Not all properties have to take part in the nesting relationship individually. They can be grouped together when the relevant information is split among multiple properties. This is particularly useful for the last layers of nesting when the remaining information does not need to be further split into groups. For example, consider a dataset of people with their first name, last name and birthday (split among year, month and day like before). One natural nesting relationship would be $year \rightarrow month \rightarrow day \rightarrow (firstname, lastname)$. We denoted the property grouping with parenthesis. This just means that after grouping by day, the data is not grouped by first nor last name but be considered as a whole. For simplicity, we consider this kind of grouping only at the end of the nesting.

For **Goal 1**, the following nesting relationship can be deduced:

$$DeliveryDay \rightarrow ItemType \rightarrow Location \rightarrow (Id, ResponsibleEmployee) \tag{3.1}$$

The clues for the first three properties are in the nested *"for each"* statements and the rest of the properties are grouped together because no other preference is stated.

Similarly, for **Goal 2**, the nesting relationship is:

$$ResponsibleEmployee \rightarrow DeliveryDay \rightarrow Location \rightarrow (Id, ItemType) \tag{3.2}$$

3.1.3 Data Modeling using Trees

We model and visualize our ontology with a tree data structure, making use of the previously described nesting relationships. Consider the generic nesting relationship:

$$P_1 \to P_2 \to ... \to P_n \to (P_n + 1, ..., P_{n+m}) \tag{3.3}$$

where $P_1...P_{n+m}$ are the properties of the dataset, more commonly referred to as the columns of the table. The first n properties are used for the nesting relationship individually, while the last m properties are grouped together.

The tree has $n + 1$ complete levels, as follows: all nodes on a level $x \leq n$corresponds to an individual property P_x, while the leaf nodes (the ones on level $n+1$) are associated with an actual item from the dataset, but having only the values of the grouped properties ($P_{n+1}...P_{n+m}$). The relationship between a parent node and a child node has a unique value associated with it (across the siblings of the child node) taken from the possible values of the parent node's associated property. These nodes and values are chosen such that we can construct a leaf node for each set of data items that have the same value for the individual relationship properties and for which the path leading from the root node to that leaf node defines property-value pairs for those first n properties, whereas the remaining m properties are associated to the leaf node itself. The general form of this tree can be seen in Figure 3.1.

As we can see, starting from the root node, the dataset can be entirely recovered and converted to its original form. The property-value pairs associated with the path leading to every leaf node combined with the data associated with the leaf node itself yield the original dataset.

Similarly, every inner node has property-value pairs associated with the path leading from the root node to that inner node. All the original data that has these property-value pairs can be retrieved using the leaf nodes from its subtree. This grouping method provides the necessary information chunking (described in Section 2.5) for the adaptive user interface (described in Section 2.3).

To create the tree, one can use a recursive grouping of the data based on the properties in the nesting relationship, as presented in Algorithm 1.

Applying this method to our example, we create the trees in Figure 3.2 for **Goal 1** and Figure 3.3 for **Goal 2**.

3.1.4 Data Selection and Sorting

To further prepare our data for visualization in the user interface, we provide one more layer of adaptation: selection and sorting of data.

Selection is done in two ways: the first one would be providing a fixed value for the first $x \leq n$ properties in the nesting relationship 3.3. Thus, the subtrees related to the other values of the first x properties are discarded. This is particularly useful when the data in the subtrees does not have to be visualized simultaneously, allow-

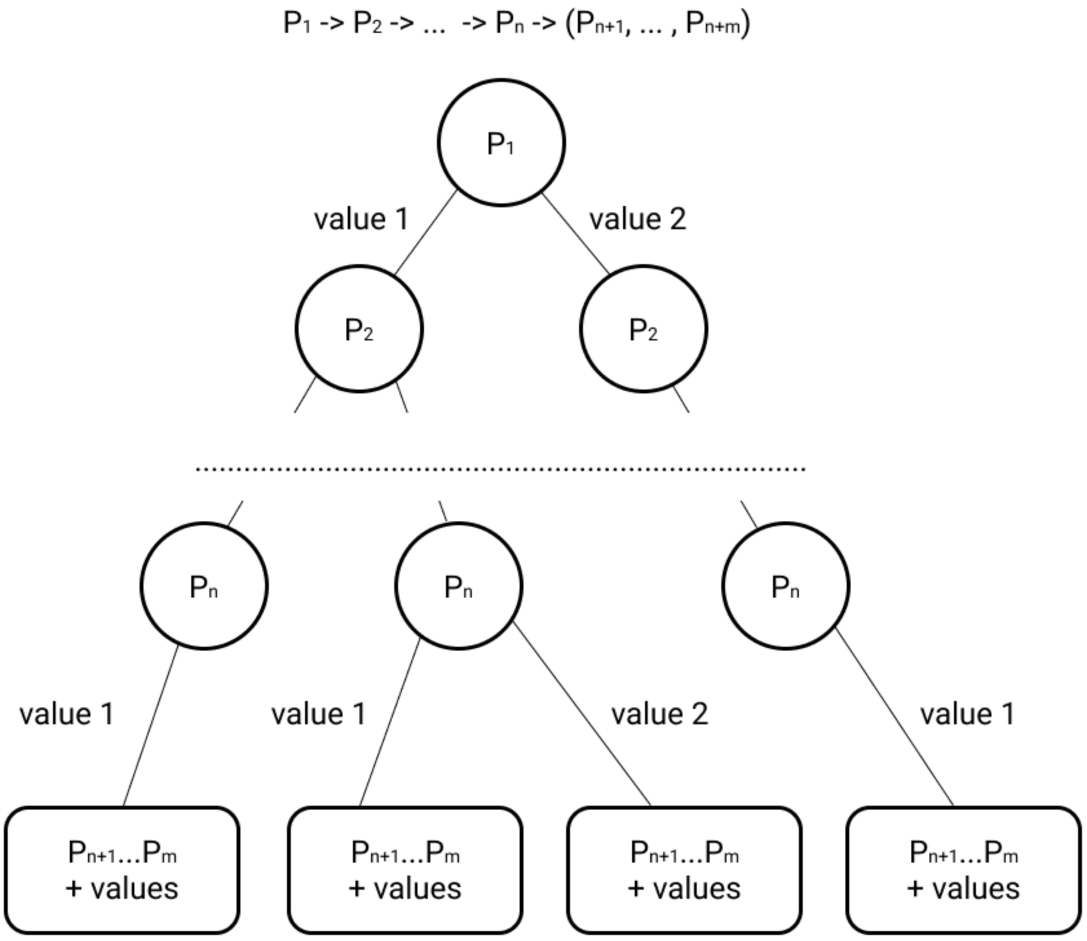

P_1 -> P_2 -> ... -> P_n -> (P_{n+1}, ... , P_{n+m})
P_1
value 1
value 2
P_2
P_2
P_n
P_n
P_n
value 1
value 1
value 2
value 1
P_{n+1}...P_m + values
P_{n+1}...P_m + values
P_{n+1}...P_m + values
P_{n+1}...P_m + values

Figure 3.1: A tree built upon the nesting relationship 3.3

Algorithm 1 Creating the tree recursively

```
function CREATETREE($data$, $nestingRelationship$)
    if $nestingRelationship$ does not have individual properties then
        return leaf node with $data$ projected on the properties in
        $nestingRelationship$ and duplicates removed
    end if
    $node$ = new Node
    $property$ = first property of $nestingRelationship$
    for each $value$ of $property$ do
        $childData$ = $data$ with $property$ having $value$
        $childRelationship$ = $nestingRelationship$ without $property$
        $childNode$ = createTree($childData$, $childRelationship$)
        Add $childNode$ to $node$ and associate $value$ to their relationship
    end for
    return $node$
end function
createTree(dataset, nestingRelationship)
```

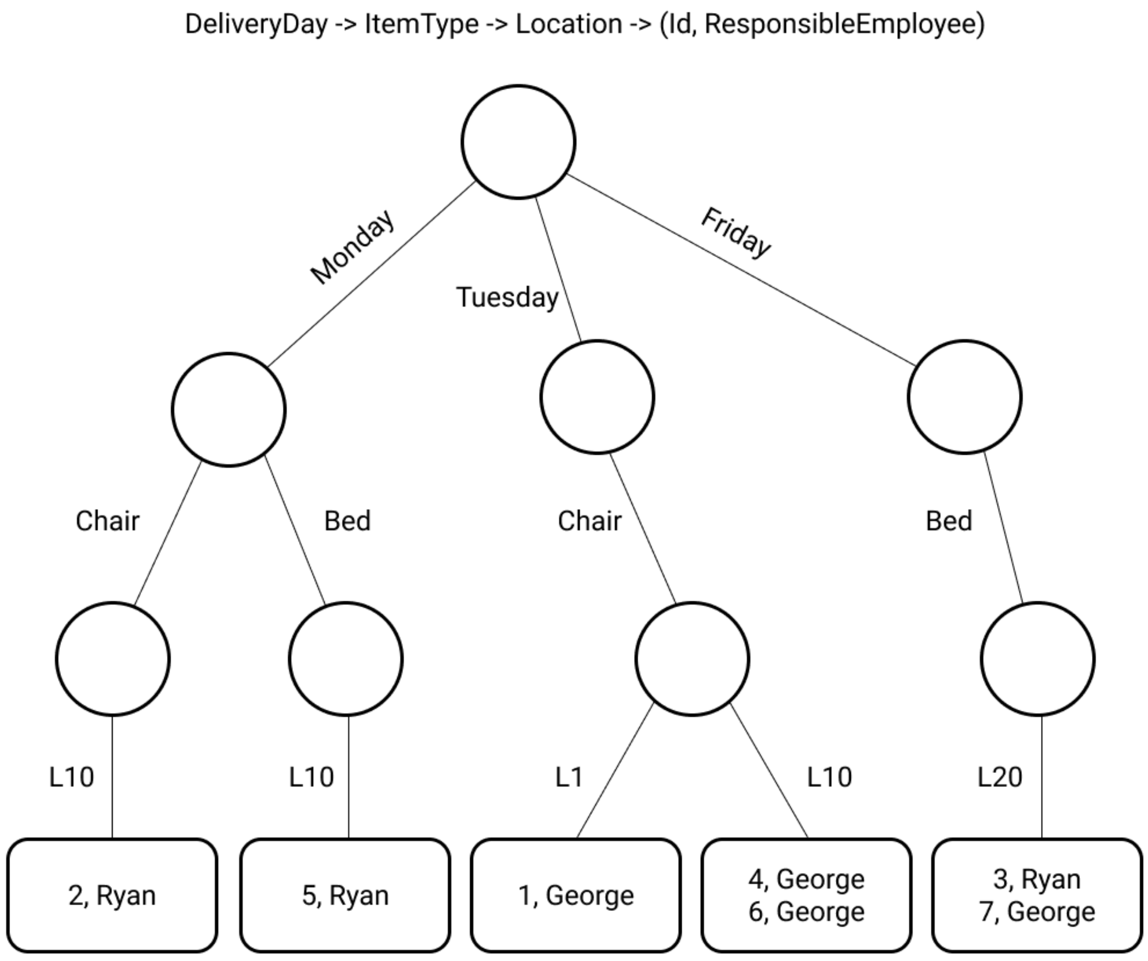


Figure 3.2: The tree build for **Goal 1** based on nesting relationship 3.1

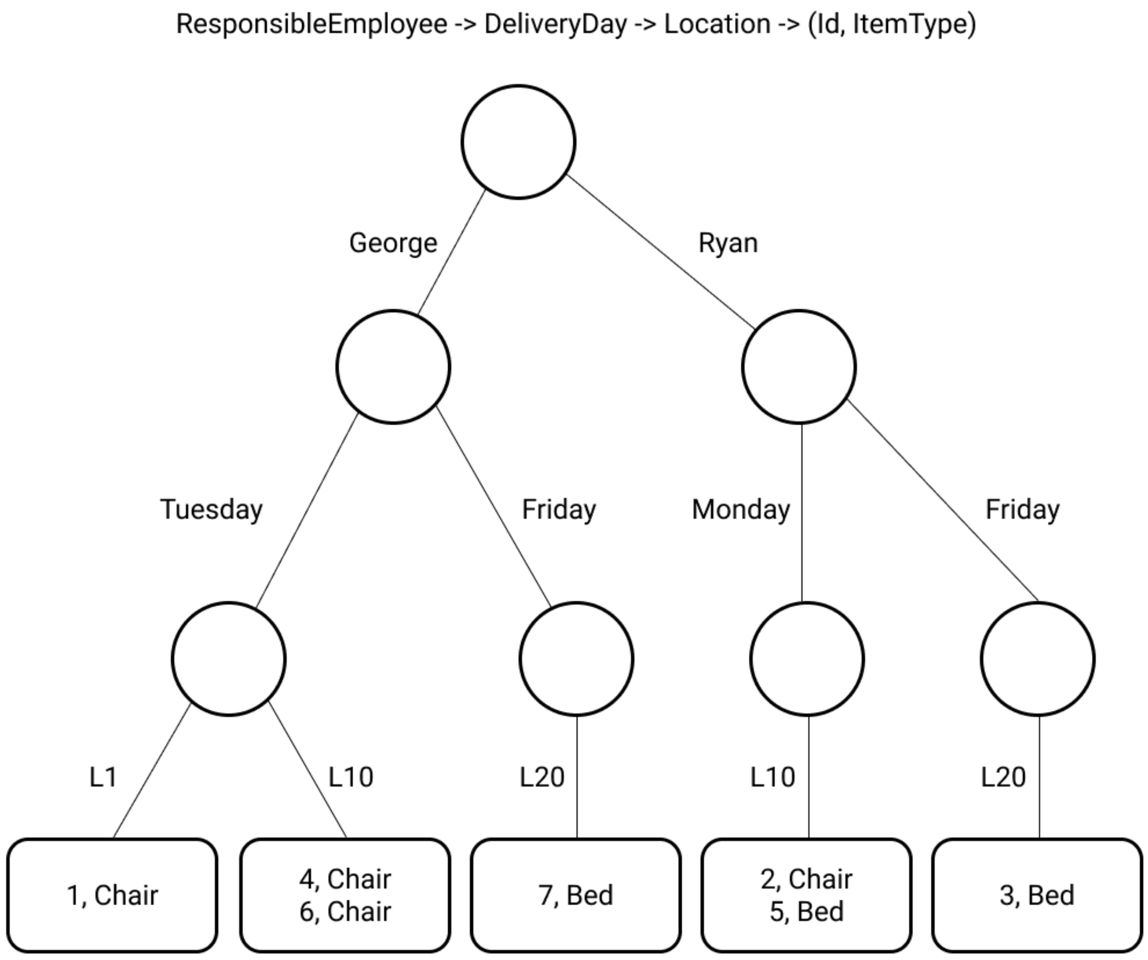


Figure 3.3: The tree build for **Goal 2** based on nesting relationship 3.2

ing the user to focus on one subset of the data at a given time. Like the *concordancer* in *"Should you be persuaded: Two samples of data-driven learning materials"* [Joh91], this is the component that filters the dataset to find the relevant entries based on the criteria chosen by the user. This selection information should be encoded in the nesting relationship, like so: $P_1^{(value=Value2)} \rightarrow P_2^{(value=Value2)}$.

The other way of making the selection would be removing properties from the right side of the nesting relationship, therefore keeping only information about the classification of the data, but not being able to reconstruct any item of the original dataset. This is used to create user interfaces that help with searching for the desired information when the exact values of the properties are not known. This type of classification allows the user to gradually select and decide upon values for properties by being presented only the valid ones (the ones that lead to data from the dataset).

Sorting is necessary for displaying the data in a consistent manner. If the order of components is not defined and changes frequently, the user cannot make use of his spatial memory to navigate the interface (see Section 2.2). Also, the ability to see the data sorted improves search times [Joh91].

We define sorting at the level of each property. The values associated with that property shall be displayed in the specified order. It can be alphabetical (ascending or descending), numerical (increasing or decreasing) or a custom order. This information is encoded in the nesting relationship for each of the properties, like so: $P_1^{(sort=asc)} \rightarrow P_2^{(sort=decr)}$. Note that for grouped properties, the actual data groups are sorted by the first property and order, and if those are equal, by the second property and order, etc.

We apply these concepts to our example's goals. Firstly, when looking at the information necessary for **Goal 1**, we may want to select some of it to inspect in more detail. Particularly, we shall select a value for the Delivery Day, and display just the subtree. This way we can focus on one delivery day at a time, instead of all at once, since we do not need to compare them in any way. Moreover, we can decide on sorting the rest of the properties alphabetically in ascending order. The resulting nesting relationship is

$$\begin{aligned} DeliveryDay^{(value=Tuesday)} \rightarrow ItemType^{(sort=asc)} \rightarrow Location^{(sort=asc)} \\ \rightarrow (Id^{(sort=asc)}, ResponsibleEmployee^{(sort=asc)}) \end{aligned} \tag{3.4}$$

and the tree representation can be seen in Figure 3.4.

For **Goal 2**, a similar approach is taken. We focus on one employee at a time, because no comparison between them is necessary. Ascending sorting for the rest of the properties is again a good default. The nesting relationship becomes

$$\begin{aligned} &ResponsibleEmployee^{(value=George)} \rightarrow DeliveryDay^{(sort=asc)} \rightarrow Location^{(sort=asc)} \\ &\rightarrow (Id^{(sort=asc)}, ItemType^{(sort=asc)}) \end{aligned} \tag{3.5}$$

and the tree representation can also be seen in Figure 3.4.

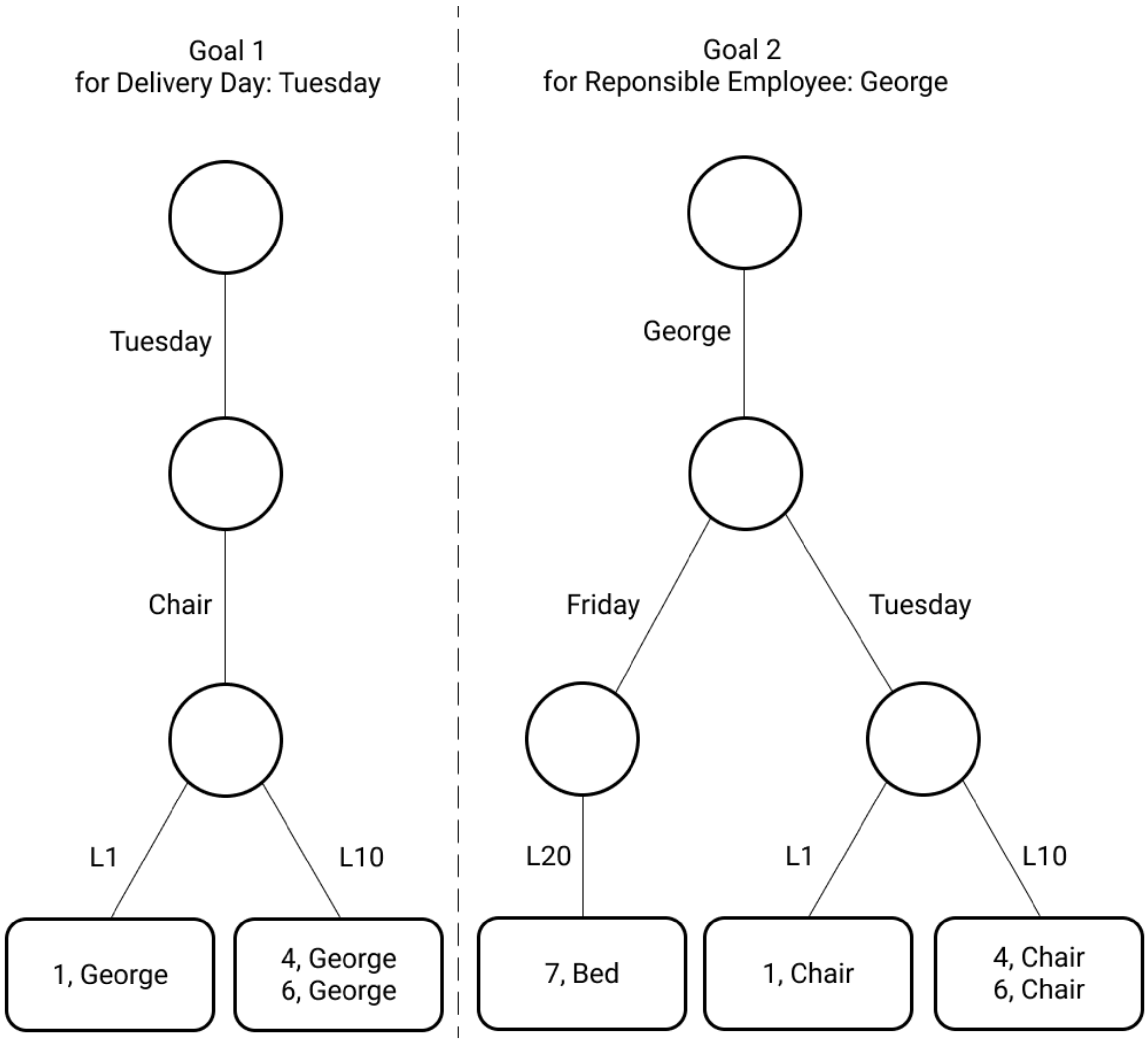


Figure 3.4: The trees built for **Goal 1** and **Goal 2** based on the selected and sorted nesting relationships 3.4 and 3.5

3.1.5 Displaying the Data

After organizing the data, the last step is to display it. As stated in *"Supporting and Exploiting Spatial Memory in User Interfaces"* [SCG13], in human-computer interfaces, knowledge of the spatial location of controls can help a user interact with the system fluidly and efficiently, without having to perform a slow visual scan. The user interface should therefore be designed to provide support for developing the user's spatial memory and it should allow the user to exploit it for rapid interaction whenever possible. We should also strive to follow the *Gestalt principles* [Prz19] and design the interface such that it supports the modeling we've developed so far.

In order to create a truly adaptive user interface, it needs to be modular. Each user interface module should be compatible with the other ones and be able to display a layer from our model while respecting the order of the values and making use of other modules for the layers below. This definition comes with the exception of the last layer, which is displayed with special modules that can group data, without needing extra modules since no layers are below it. As such, a whole subtree can be represented with a number of modules equal to the number of layers.

Since the modules are built to be flexible and interchangeable, the user can assign these modules to the properties (which are themselves bound to the model layers) in any way he desires. It is worth noting that properties with a specified (fixed) value do not need to be assigned a module, but they should all be displayed on the screen to clear ambiguity. Module assignments can also be specified in the nesting relationship: $P_1^{(module=1)} \rightarrow P_2^{(module=2)}$. With this final enhancement, the nesting relationship becomes the adaptive user interface configuration that can be freely customized by the user. The system adapts to these preferences by modeling the data in the dataset, performing the necessary selections and sorting, attaching the data to the interface modules and finally displaying it in an interactive manner.

As a guideline for the modules, each one is responsible for rendering the interface for a subtree in the data model. Since each module is bound to a property, each possible value of this property is displayed in the order specified by the user, along with a container for the module on the next level. This module is, in turn, responsible for rendering the model subtree associated with that value. These contained modules need to have a clear separation between them to help the user perceive the information chunks. Examples can be seen in Figure 3.5.

To finally show how we achieved the initial example's two goals, we make use of the example modules in Figure 3.5 to create user interfaces. For **Goal 1**, we attach modules in the following way:

$$\begin{aligned} DeliveryDay^{(value=Tuesday)} \rightarrow ItemType^{(sort=asc,module=1)} \rightarrow Location^{(sort=asc,module=2)} \\ \rightarrow (Id^{(sort=asc)}, ResponsibleEmployee^{(sort=asc)})^{(module=3)} \end{aligned} \tag{3.6}$$

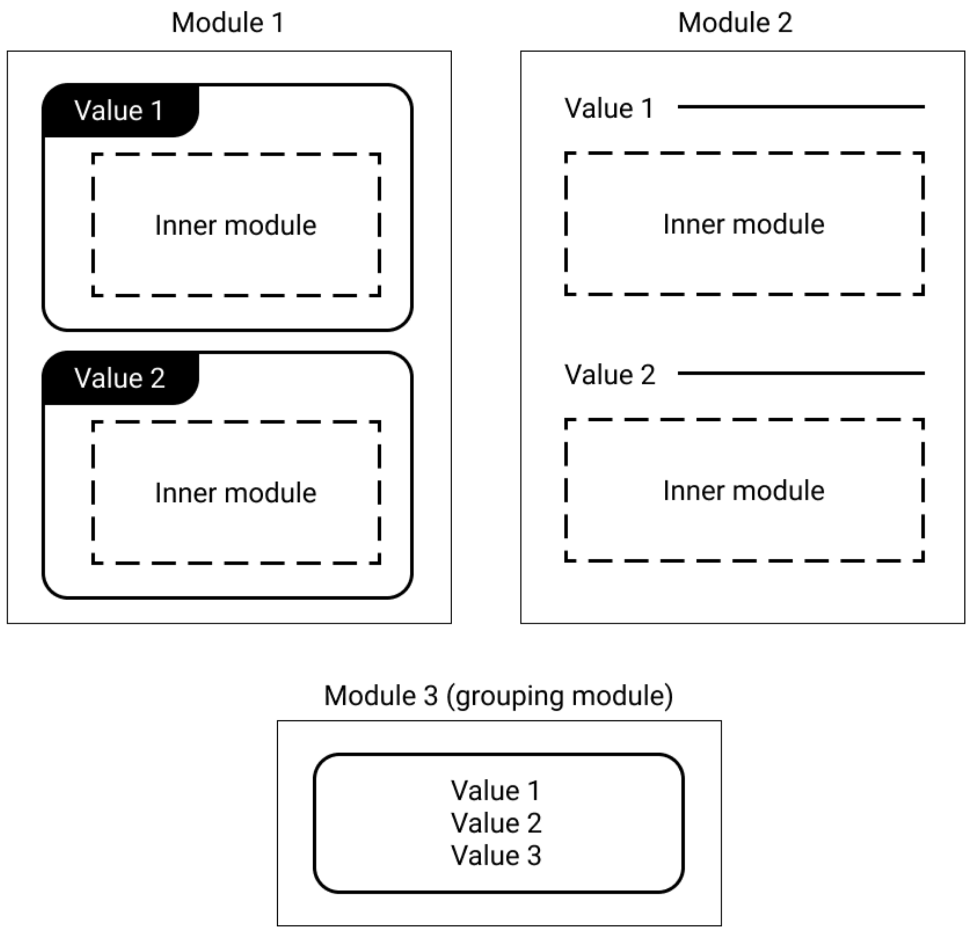


Figure 3.5: Examples of modules for the adaptive user interface

and similarly, for **Goal 2**:

$$\begin{aligned} &ResponsibleEmployee^{(value=George,module=1)} \rightarrow DeliveryDay^{(sort=asc,module=2)} \\ &\quad \rightarrow Location^{(sort=asc)} \rightarrow (Id^{(sort=asc)}, ItemType^{(sort=asc)})^{module=3} \end{aligned} \tag{3.7}$$

Both adaptive user interfaces constructed for these goals can be seen in Figure 3.6.

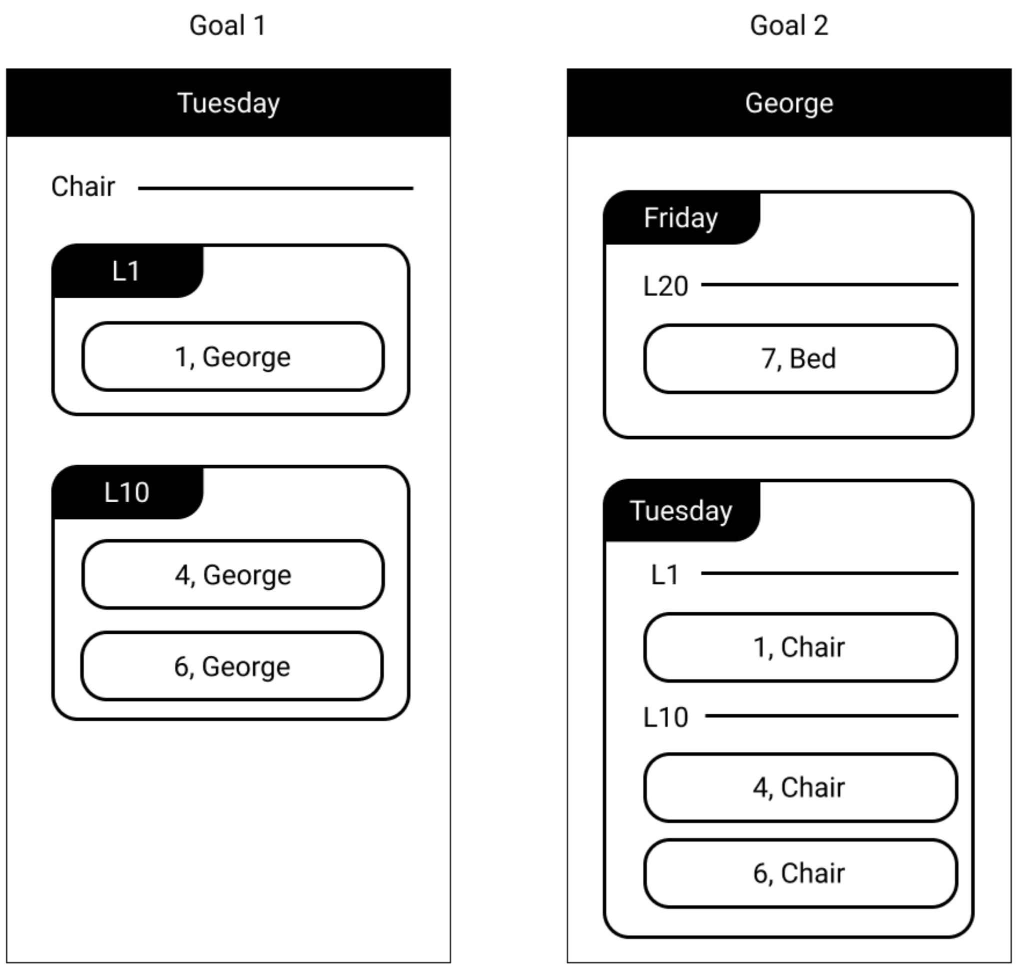


Figure 3.6: Final user interfaces for **Goal 1** and **Goal 2** built upon configurations 3.6 and 3.7

3.2 Requirements & Specification

3.2.1 Overview of the System

There will be two types of users for this system: the administrators and the normal users. The aim of the system is for the normal users to make decisions based on tabular data provided by the administrators using the method proposed in Section 3.1.

The administrators will upload the datasets on their computer, which will be stored in a centralized database and all the normal users will be able to access them from their mobile devices. The user interface for the mobile devices must adapt to the user's preferences in order to facilitate various kinds of decisions that must be taken based on the provided data.

Therefore, the system will be composed of a mobile application, a desktop application and a server application, whose requirements and specification will be described in the following sections.

3.2.2 Desktop Application

The administrators will interact with the system via an application on their desktop computer. Since the administrators have different kinds of operating systems, the application is required to run on most of them, including Linux, macOS and Windows.

After providing the server connection details and logging in by providing valid credentials (username and password), the administrator will be able to create multiple datasets. With each dataset, he will associate a name and a color for quick identification. The dataset can be defined by uploading a CSV (Comma Separated Values) file that contains the tabular data along with the names of the columns (properties of the data items) on its first row. This way, data can be imported from several systems that can export tabular data, for example relational databases.

An administrator can see all the dataset definitions and is able to delete a dataset he created as well as re-upload a CSV file to update it. The new CSV file must have the same column names as the previous ones in order to be considered a valid update. Normal users can then use a button to update their interface with the new data.

When viewing a dataset, an administrator can see the name, color and properties of the dataset as well as its size. All the actions will be available only when the application is connected to the server.

The application needs to be straightforward to install and run, have good performance, notify the user about its background operations and display all encountered errors in a friendly format.

3.2.3 Server Application

The server application will mediate the connection between the mobile application, desktop application and database. The interfacing with the two user applications will be done via an HTTP API (Application Programming Interface), whereas the communication with the database will be done through its specific driver.

In the API intended for the **desktop** application, the operations specified in Section 3.2.2 should be possible:

- Authenticating
- Creating, updating and deleting datasets
- Retrieving information about datasets (properties, name, color, size, creator)

The operations that are specific to administrators need to be secured by credentials (defined at system initialization) and not publicly accessible.

In the API intended for the **mobile** application, the necessary operations are:

- Retrieving information about the datasets (name, color and properties of the data)
- Retrieving the subtree described by a given nesting relationship

All these operations will be publicly accessible.

The server application needs to be easy to deploy and configure. It also has to be able to handle simultaneous connections and serve updates in real-time. Optimizations must be made to facilitate low response times when similar classification requests are made.

3.2.4 Mobile Application

The normal users will interact with the system via an application on their Android smartphone. After they provide the server connection details, they will be able to see a list of the available datasets and their associated name and color. After selecting one of them, they will be able to configure the interface for data visualization.

The configuration is based on the method described in Section 3.1 and consists of incrementally defining a nesting relationship comprised of:

- **Valued properties** - properties that are associated with a fixed value
- **Classification properties** - properties that are associated with an interface module
- **Grouped properties** - properties that are displayed together in a grouping module

For each of them, a sorting order might be provided or otherwise it defaults to ascending order. Also, for the valued properties, the application will provide suggestions for the values by querying the server. Equation 3.8 represents the kind of partial relationship used by this process for a dataset with n properties, configuring property $x \leq n$.

$$P_1^{(value=Value1)} \rightarrow P_2^{(value=Value2)} \rightarrow ... \rightarrow P_x \tag{3.8}$$

Using this relationship, only the possible values for the selected property are displayed and thus the user will not be able to choose one that results in an empty subset of data.

After the user is satisfied with the configuration, the application will display a preview of the interface using sample data, having the option to save the configuration, download the classified data in the internal memory and show the actual generated user interface. If the user needs to change the interface, he will wait for an internet connection and make the configuration again, according to his needs. If he instead only desires to update the interface with new data but preserve the configuration, he will be able to do this using a dedicated button.

Since the data and configuration are stored in memory, the generated user interface should survive restarts and other changes in the application lifecycle, such as screen rotations or switching between applications. Also, the user should have the option to visualize the properties that he chose for each interface module.

The application should be easy to install and navigate as well as have good response times for basic actions such as scrolling or pressing buttons. It should display loading animations whenever a background process is ongoing and the interface should not freeze. Error messages will be displayed in a friendly format and will not disrupt the user's experience.

3.3 Design & Implementation

This sections presents the architectural details of the system implementation.

3.3.1 Desktop Application

The desktop application is built in Kotlin [Section 3.4.2] and its architecture [Figure 3.7] follows the MVVM architecture [Hal11] with its main three components: Model, View and ViewModel.

Below, we describe these components along with the interaction between them.

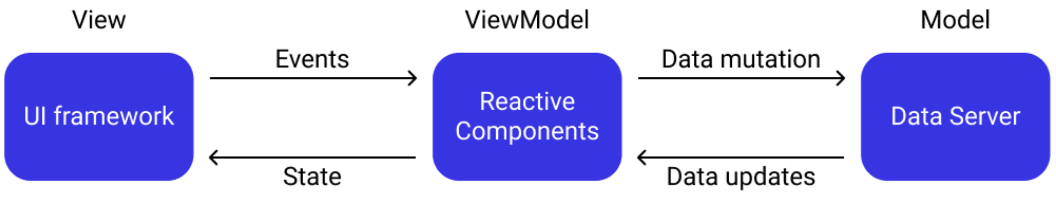


Figure 3.7: Desktop application architecture based on MVVM

- **Model** - The model represents the business logic or back-end of the application, where data processing and storage takes place. In our case, where the application only works while online, the server application [Section 3.3.2] will fulfill this role.
- **View** - The view represents the component that interacts directly with the user by displaying data on the screen and capturing his input. The UI framework used for this component's implementation is Compose for Desktop [Section 3.4.6].
- **ViewModel** - The ViewModel component is responsible for managing the interaction between the model and the view using reactive components such as Mutable State objects [Section 3.4.6] and Kotlin Flows [Section 3.4.3]. User interaction with the interface triggers events that are transmitted to the ViewModel through normal function calls which mutate the internal state accordingly (mainly regarding the display of loading bars). They also asynchronously send HTTP requests using Ktor Client [Section 3.4.7] to mutate the data on the server, which will, in turn, send the updates back through the WebSocket. Data is received from this WebSocket and pipelined through reactive streams, then sent to the view component responsible for displaying it in the user interface. This data flow respects the Single Source of Truth (SSOT) principle [PS14] and therefore avoids several possible inconsistencies.

3.3.2 Server Application

The server application is written in Kotlin [Section 3.4.2] using the Ktor Framework [Section 3.4.7] to provide an HTTP API that is consumed by the other two applications. Since the data that we store does not have a fixed structure and is required to transform into tree representations, the server uses a graph database, Neo4J [Section 3.4.8]. There is no schema for this kind of database, but we give a few examples and explanations for the data management strategy.

Figure 3.8 presents the graph produced when the user *admin* (the orange node) creates the *Warehouse* dataset. The dataset node (displayed in blue) contains all the meta information about the dataset, such as *name*, *color* and *properties*. Every dataset contains a number of items (displayed in red), each of them having the appropriate values for the properties. Only the *id* of the item is displayed in our figure for brevity.

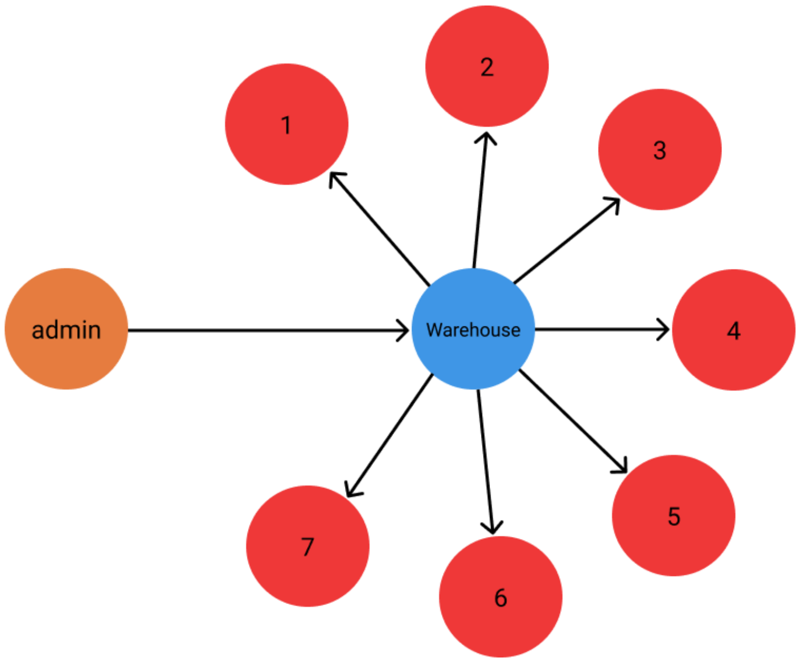


Figure 3.8: Visual representation of the graph database storing a dataset with its items and user that created it

Whenever a classification request is received, the server application clusters the data items and creates the necessary nodes and relationships. Since these remain in the database, subsequent requests with similar nesting relationships will be handled faster, only a simple graph navigation being enough to obtain the data. An example of this classification can be seen in Figure 3.9 where the *Warehouse* dataset is classified by its *Item type* property (represented by the green node). The two possible values are represented by purple nodes and each of them is linked with the appropriate *items*, the ones which have that value for the *Item type* property. The presented process occurs recursively for all the items in the nesting relationship since every set of data items can be further classified.

Note that Figure 3.9 is not an exhaustive representation of the stored graph: the edges and user node represented in Figure 3.8 have been omitted for conciseness. There is a single graph that holds the information for all computed classifications at the same time.

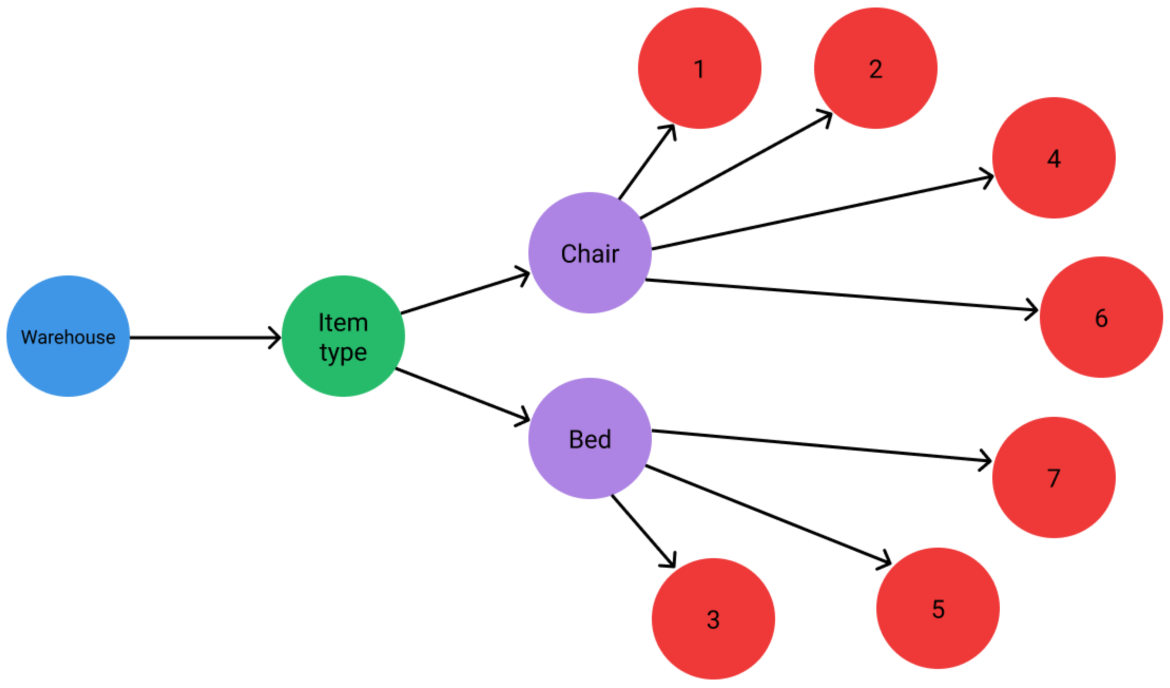


Figure 3.9: Graph representation of the dataset being classified by its *Item type* property

The server and database applications each run in a separate Docker container [Section 3.4.9] in order to be easily deployed and maintained.

3.3.3 Mobile Application

The mobile application is built for Android [Section 3.4.4] in Kotlin and its architecture [Figure 3.10] also follows the MVVM pattern.

Below we describe the components along with the interaction between them.

- **Model** - As before, the model represents the back-end of the application, where data processing and storage takes place. In our case, the server application [Section 3.3.2] will provide the datasets and the application's local storage will act as a repository for the chosen items and the nesting relationship configuration specified by the user. Furthermore, it will emit signals when any of the data changes so that the other components can react accordingly. Datastore [Section 3.4.5] is the appropriate library for this situation since it can store our data in the device's memory and notify the system when it gets updated.

- **View** - The view represents the component that displays data on the screen and captures the user's input. The UI framework used for this component's implementation is Jetpack Compose [Section 3.4.6] since it uses a declarative

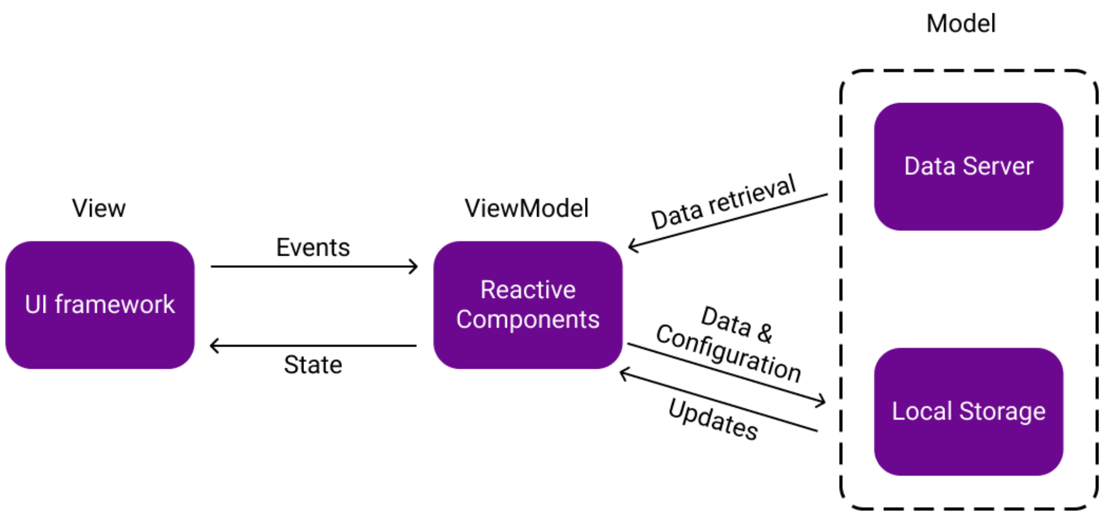


Figure 3.10: Mobile application architecture based on MVVM

approach for creating the user interface and therefore provides excellent support for our modular approach.

- **ViewModel** - As previously stated, the ViewModel is responsible for managing the interaction between the model and the view using reactive components such as Mutable State objects and Kotlin Flows. User interactions with the interface triggers events that are transmitted to the ViewModel through function calls. These events range from choosing items from a list to configuring the nesting relationship used for adapting the user interface. Effects of these events include mutating the internal state (for showing messages and progress) and asynchronously sending HTTP requests using Ktor Client to retrieve data from the server. The ViewModel also receives updates from the local storage and reacts accordingly by regenerating the state and sending it to the view for adapting the user interface. Again, the Single Source of Truth (SSOT) principle is used to avoid displaying inconsistent data.

3.4 Technologies Used

This section presents the main technologies used for implementing the system.

3.4.1 Java Platform

The Java Platform [GM95] was initially created by James Gosling at Sun Microsystems and provides a means to develop software and deploy it in a crossplatform environment. The software, usually written in the Java language, is compiled into Java bytecode that can be run on a Java Virtual Machine, whose implementation differs across different device architectures but follows the same specification for all of them, resulting in identical execution of the bytecode. Therefore, the motto of the platform is *Write once, Run anywhere*.

We chose to develop all three applications for the Java platform mainly because of the flexibility and stability it provides for all three environments (mobile devices, desktop computers and servers).

3.4.2 Kotlin

Kotlin [SB17] is a programming language developed by JetBrains that is known for its concise but explicit syntax, interoperability with the Java platform and DSL capabilities, to name a few. The language has an increasingly large community that contributes in different ways to its evolution. Since 2019, Google has announced that Kotlin is the preferred language for developing Android applications and has provided developers with special libraries and tools for it. Also, JetBrains has created Kotlin Multiplatform, a set of tools that facilitate code sharing between multiple platforms: Kotlin for Java Virtual Machine, Kotlin for JavaScript and Kotlin for Native applications.

The Kotlin language was used to develop all three applications (mobile, desktop and server) especially because of its interoperability with the Java platform and the other chosen technologies (Kotlin Coroutines and Flows, Compose and DataStore).

3.4.3 Kotlin Coroutines and Flows

One of the novelties of the Kotlin programming language is the way it manages asynchronous and reactive programming. Coroutines [Eli18] are often referred to as lightweight threads, and their implementation in Kotlin has a concise syntax that appeals to programmers. Moreover, the Kotlin implementation supports structured concurrency, which eliminates exception leaks and unclosed resources. Also, Kotlin facilitates reactive programming with Kotlin Flows [Eli19], an original method to handle cold streams using coroutines, making use of all their advantages.

Since our system heavily relies on asynchronous execution of code for handling events such as user interactions and changes to underlying data, these technologies provide the necessary safety needed while handling the multi-threaded execution.

3.4.4 Android

Android [Gar11] is an open-source smartphone operating system platform based on Linux that uses the Java Platform for the applications running on it. It is developed and maintained by Google, although the companies developing smartphones usually create a modified version to ship with the manufactured devices. Developing of applications is done using the Android SDK and most of the times the Android Studio IDE, making use of Gradle for building and running the applications as well as resolving dependencies of projects.

The platform is suitable for our needs since it is compatible with the rest of the chosen tools, widely used and imposes few limitations for application development (operating system or hardware).

3.4.5 DataStore

Android Jetpack DataStore [Kan20] is an architectural Android library developed by Google that facilitates easy access to a device's local storage. It comes in two flavors: type unsafe key-value storage and typesafe custom data storage. Both of these variants provide an API for asynchronously writing data using Coroutines and listening for changes asynchronously via Flows.

Although currently in its Beta development stage, the typesafe library is a perfect fit for the mobile application's local data storage since it integrates excellently with the chosen asynchronous technologies.

3.4.6 Compose

Compose is a user interface library that facilitates the creation of interfaces using declarative style code. The aim of the tool is to build components that react to the mutation of application state by adapting the interface to reflect the changes. Google and JetBrains collaborated for building it and ended up with two versions: **Jetpack Compose** [Ric20], the library maintained by Google for the Android Platform and **Compose for Desktop** [Frä21], the library maintained by JetBrains for desktop applications. Common code can be shared between the two platforms using Kotlin Multiplatform. The library uses a compiler plugin that interprets the code differently, translating Composable functions into interface components.

Although at the time of writing, these technologies are still in their Beta development stage, their usage highly reduces the workload of creating the adaptive user interface modules. The declarative approach of the library perfectly matches the style of our proposed method, the ontology and data guiding the creation of the interface.

3.4.7 Ktor

Ktor [Mor19] is a Kotlin/JVM asynchronous framework created and maintained by JetBrains that facilitates network communication. It comes in two versions, Ktor Client for making HTTP requests over the internet and Ktor Server for handling requests. The software is modular, meaning it allows the installation of various modules that suit different needs and are individually configurable.

In our implementation we use Ktor Client in the mobile and desktop applications and Ktor Server in the server application in order to make the communication between the software possible. The main modules utilized are the ones that handle request routing, WebSockets, authentication, authorization and data conversion.

3.4.8 Neo4J

Neo4J (Network Exploration and Optimization 4 Java) [VWA+15] is a graph database management system developed by Neo4J Inc. It is transactional, ACID-compliant and has native graph storage and processing. Its community edition is open-source, but some extensions for it are under a closed-source commercial license. As the name suggests, the system is implemented in Java, but the mutation and retrieval of data are done using the Cypher query language through an HTTP endpoint or the binary *bolt* protocol.

Since our method heavily relies on data modeling using tree data structures, a graph database is the ideal choice for the storage and management of schemaless data and the models that are constructed based on it.

3.4.9 Docker

Docker [Tur14] is a Platform-as-a-Service (PaaS) set of tools that create OS-level virtualization in the form of containers. These containers are more lightweight than virtual machines and can host applications and services along with their dependencies in an isolated manner, communicating through well-defined channels with other containers. The creation of container images (blueprints for containers) is done using a Dockerfile that defines the incremental steps necessary to mutate the chosen initial virtualized operating system image into the custom one with the configured application. After an image is created, it can be used to create and run containers. Since the tools are highly optimized, the creation and deletion of containers is very fast compared to the same operations for a standard virtual machine and thus greatly improve the workflow, be it testing or running a production application.

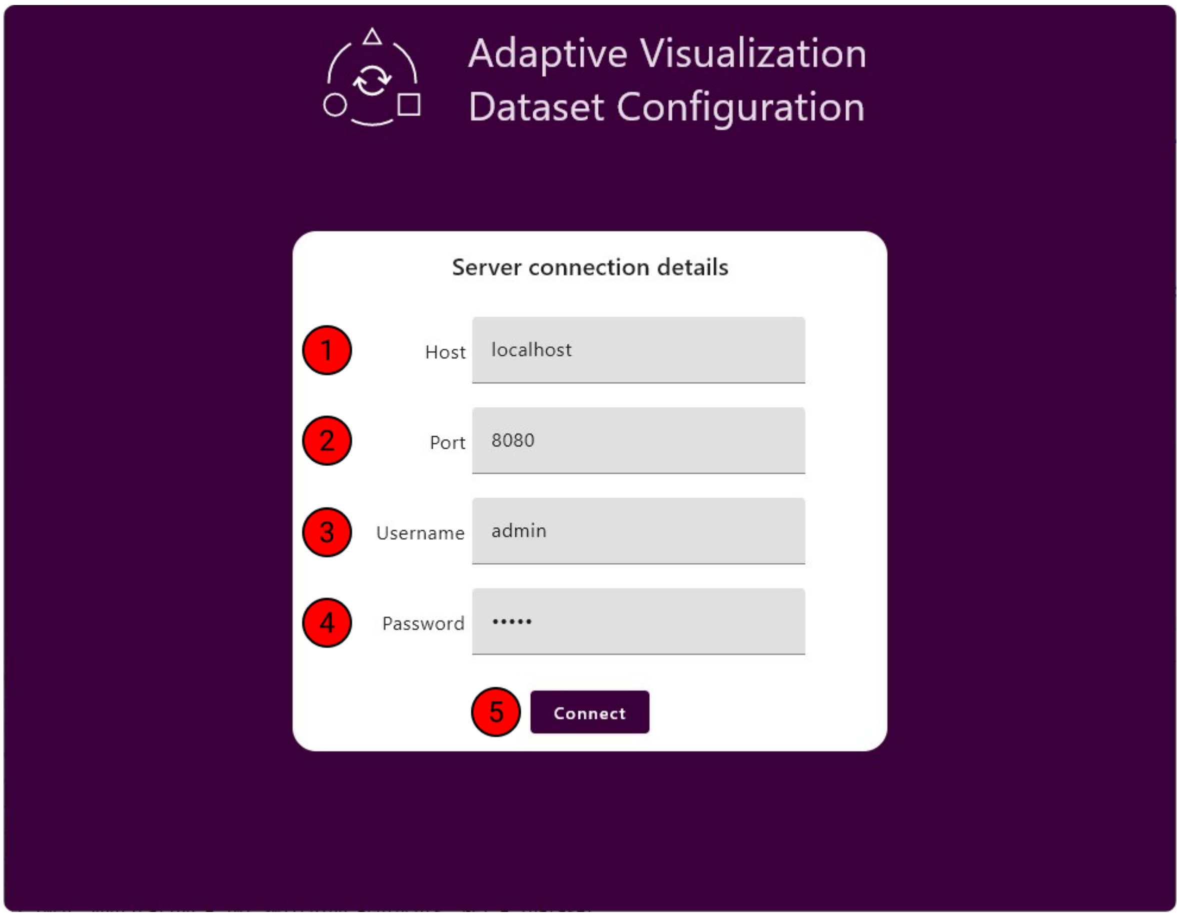
Our server and database management system need to be efficiently deployed on a production server and securely isolated from other processes running on it.

Making use of Docker images addresses those concerns and enables us to test the application in different environments faster.

3.5 User Interface Guide

3.5.1 Desktop Application

After installing and executing the application, the user is presented with the Login page [Figure 3.11].


Adaptive Visualization
Dataset Configuration

Server connection details

Host	localhost
Port	8080
Username	admin
Password	•••••

Figure 3.11: Desktop application Login page

The aim of this page is to allow the user to provide the appropriate details for connecting to the data server application. The numbered controls are described below.

1. **Host** - The host of the server.
2. **Port** - The port of the server.
3. **Username** - The username for the user's account.

4. **Password** - The password corresponding to the specified username.

5. **Connect button** - This button attempts to connect the application to the server using the specified details and navigates to the Main page on success.

Once connected to the server, the Main page [Figure 3.12] presents all the necessary information and allows the user to perform the needed actions.

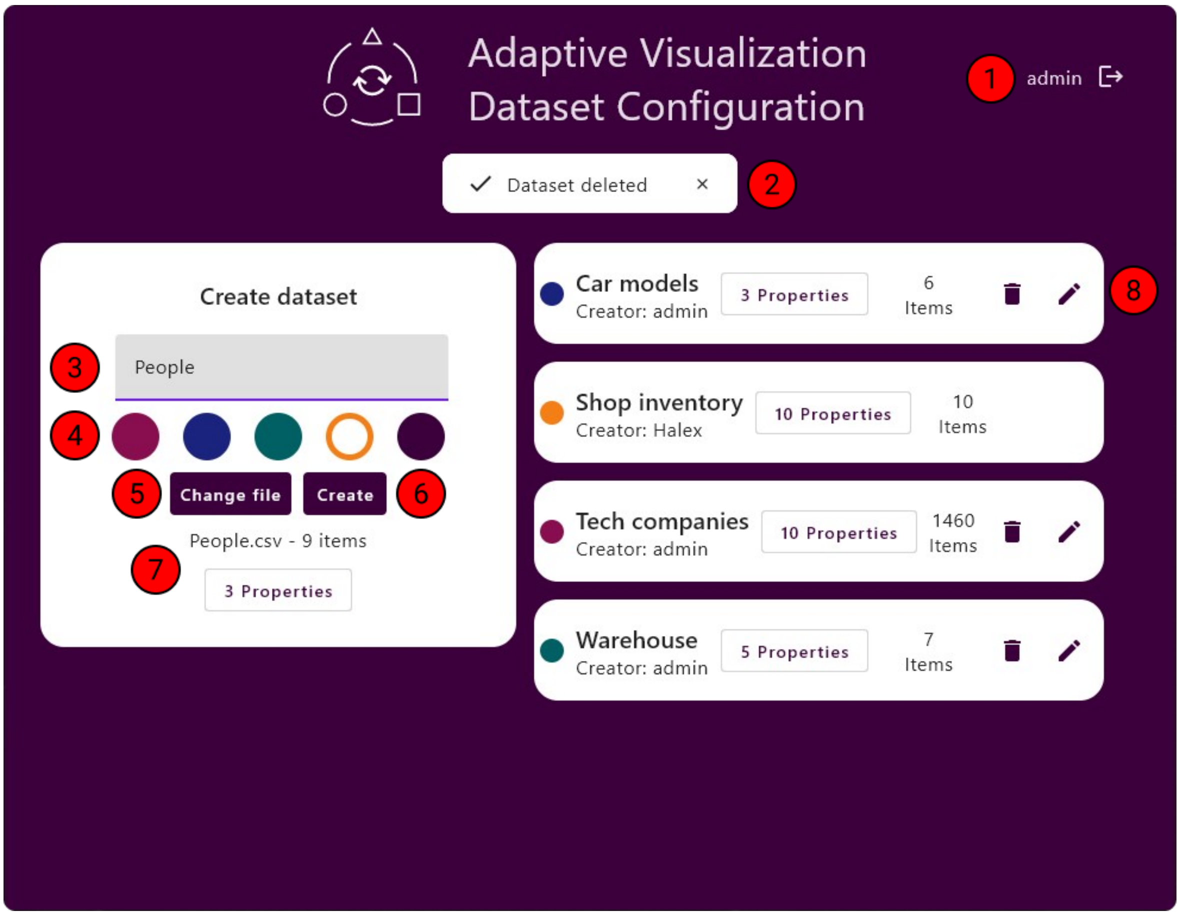


Figure 3.12: Desktop application Main page

The numbered controls are described below.

1. **Account area** - The Name of the current username is displayed. When the button on the right is pressed, the user logs out and returns to the Login page.

2. **Message area** - All the messages for the user will be displayed in this area. The messages disappear automatically after 5 seconds but can also be dismissed manually by pressing the small *x* on the right side.

3. **Dataset name** - This will be the name of the dataset being created.

4. **Color** - The user may pick a color for the new dataset by pressing on it. The selected color bullet is filled with white.

5. **Choose/Change file button** - This button can be used to select the CSV file from where to import the dataset. A file picker will be opened where the user can browse their file system and pick a file of their choice.

6. **Create button** - The button creates the dataset with the chosen properties.

7. **Selected file information** - The selected file (if any) has its name, number of items and number of properties displayed here. The properties can be seen by pressing the button.

8. **Dataset list** - All the datasets in the system are displayed in a list. For each one, the application displays the name, color, user that created it, number of properties and number of items in the dataset. The properties can be seen by pressing the button. If the current user created the dataset, the trash and pencil buttons are also displayed and can be used to delete the dataset and update it by providing a new CSV file.

3.5.2 Mobile Application

After installing and running the application, the user is presented with the Main page [Figure 3.13]. On this page, the currently configured interface will be displayed along with the name of the original dataset and a set of buttons:

- **Settings button** - This button navigates the user to the settings page, where the interface can be reconfigured.

- **Refresh button** - When pressed, this button will attempt to fetch new data from the server and update the interface accordingly.

- **Help button** - This button toggles the help mode, which gives visual hints about the placement of each properties' values and also shows the fixed values, previously hidden for brevity.

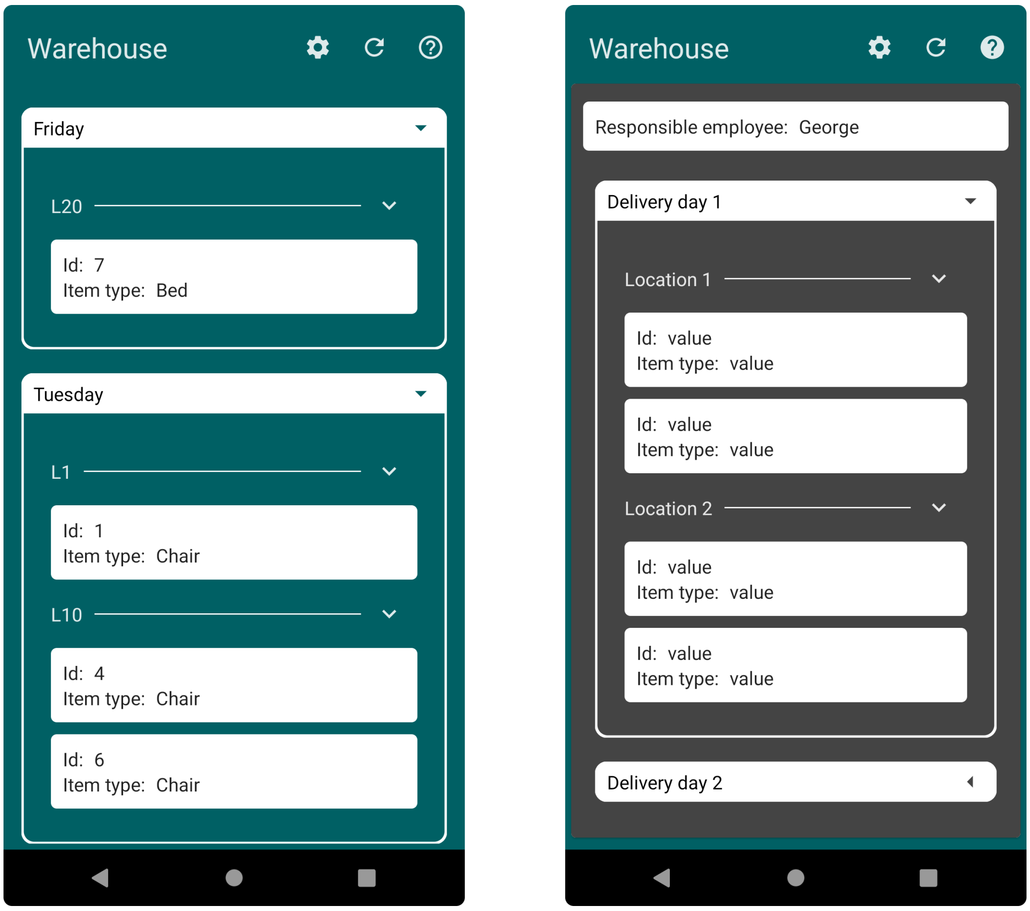
In the Settings page [Figure 3.14] the user can configure the generated interface according to his needs by following these steps:

1. Fill in the URL of the server that holds the datasets and press the arrow button.

2. Choose the desired dataset from the list.

3. Configure valued properties if needed. Upon selecting a property, the possible values are fetched from the server and displayed in the chosen order. Already configured valued properties are displayed and can be deleted.

4. Configure classification properties by selecting a property, sorting order and the associated graphical module. The properties will be displayed, with the possibility of changing their order or removing them completely.
5. Configure grouped properties by selecting them individually or adding all of the remaining ones. The selected properties are displayed in a list. Their order can be changed or they can be removed.
6. Preview the result and download the data in order to generate the user interface. This action will redirect the user to the Main page and save the configuration and data to the device's internal memory.


Warehouse

Friday

L20

Id: 7
Item type: Bed

Tuesday

L1

Id: 1
Item type: Chair

L10

Id: 4
Item type: Chair

Id: 6
Item type: Chair

Warehouse

Responsible employee: George

Delivery day 1

Location 1

Id: value
Item type: value

Id: value
Item type: value

Location 2

Id: value
Item type: value

Id: value
Item type: value

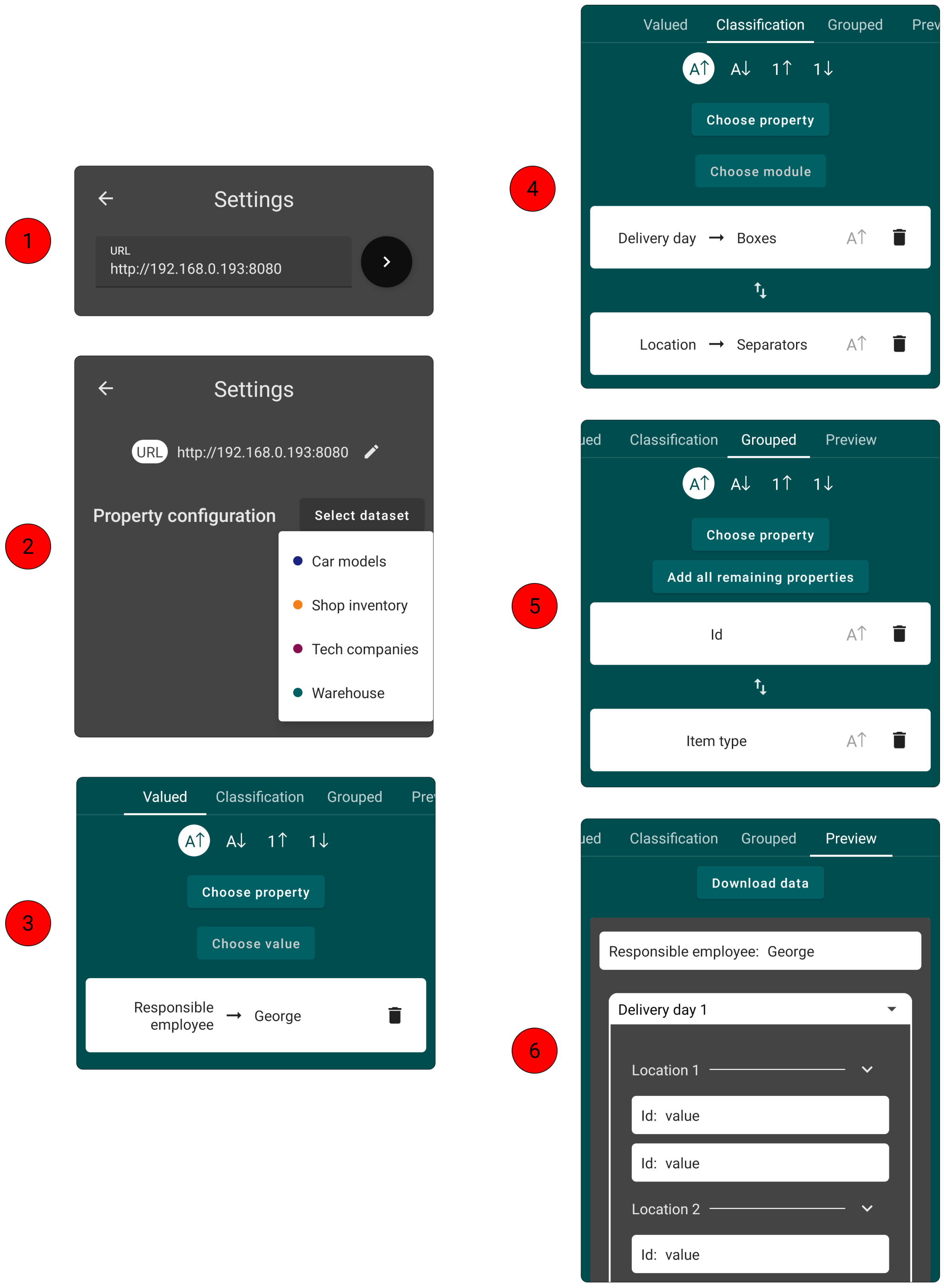
Delivery day 2

Figure 3.13: Mobile application Main page (generated interface and help mode)

3.6 Validation & Verification

3.6.1 Method Validation

Validation of the method was done through an experiment, conducted with the help of 4 people that received the mobile application. In order to test the influence



URL
http://192.168.0.193:8080

URL http://192.168.0.193:8080

Responsible employee → George

Delivery day → Boxes

Location → Separators

Id

Item type

Responsible employee: George

Delivery day 1

Location 1

Id: value

Id: value

Location 2

Id: value

Figure 3.14: Mobile application Settings page (by steps)

that domain knowledge has on the decision-making process, the participants were equally split into two groups and assigned a dataset as follows:

- Participants from group **A** were also given access to the desktop application and were asked to upload their own dataset, preferably filled with data that was related to their interests or knowledge domain.
- Participants from group **B** were each uniquely assigned a dataset uploaded by a participant from group **A**, which did not coincide with their interests or knowledge domain.

Using the mobile application with its adaptive interface, the participants from both groups were asked to make a predetermined decision based on their assigned dataset in 3 stages:

1. The dataset could only be seen as a list of items, with no grouping or filtering.
2. The dataset was displayed with the adaptive interface, but with a fixed interface configuration.
3. The dataset could be visualized using a custom configuration of the user interface, defined freely by the participant.

In order to compare the efficiency of the decision-making processes, the duration of each stage was measured for all participants.

Since the groups are randomly split, there is a possibility that a group performs worse than the other because of factors unrelated to the experiment. To overcome this issue, the experiment was conducted once with the initial groups and then repeated after interchanging them. The results are documented in Table 3.2.

Nr	Group	Dataset	List of items	Fixed configuration	Custom configuration
1	A	1	15.3	5.5	1.9
2	A	2	168.3	71.0	23.7
3	A	3	245.8	84.9	32.1
4	A	4	45.6	22.5	14.3
5	B	1	20.4	7.2	3.1
6	B	2	214.0	115.0	26.7
7	B	3	290.6	99.2	47.8
8	B	4	68.9	26.2	16.2

Table 3.2: The time required by the participants to make each decision, measured in seconds

Because the sample size is small, we are not able to prove the effectiveness of our method. Therefore, the results only give hints about its validity: making decisions based solely on the list of data items took the most time in every situation, while the inferences drawn using a customized configuration of the user interface took the least time. Also, group **A** performed better than group **B**, which might indicate that the expertise effect [SWD+20] takes place.

At the end of the experiment, the participants were asked to provide feedback. The conclusions from the feedback session are:

- The participants would benefit from having adaptive interfaces in some data-driven applications that they use on a daily basis such as the ones managing events, emails or shopping lists.
- The preview of the interface proved very helpful when creating the configuration.
- Filtering based on complex criteria would be beneficial for several situations.
- Custom sorting orders could help significantly in a few cases, such as displaying weekdays.

3.6.2 System Verification

During the development of the system, we performed several kinds of manual tests:

- Unit tests for all the components in the system.
- Integration tests for the components within an application.
- Integration tests for the system as a whole.

Most of the frameworks and off-the-shelf components used were thoroughly tested by their creators and thus decrease the chance of problems arising in our system. Unfortunately, the same cannot be stated for the technologies in the Beta stage of development, where faults and incorrect behavior are highly probable. Since we are not able to extensively test these technologies, we closely watched the issue reports for them and avoided possibly erroneous components.

Also, while conducting the experiment from Section 3.6.1, the system behaved as intended with no data inconsistencies, server downtime or application crashes.

Considering all the testing performed and precautions taken, we can safely assume that the system functions correctly in normal circumstances.

Chapter 4

Conclusions & Future Work

The chapter presents the main conclusions of the thesis and possible future work.

4.1 General conclusions

The aim of this thesis was to provide an original data-driven solution for the impediment created by large amounts of data items with fixed structure in the context of decision making. Chapter 1 presented the issue and the motivation for solving it. Chapter 2 presented in-depth the context of the issue and the related work in the domain. Finally, Chapter 3 described in detail the proposed solution, a possible implementation for it and its validation.

The solution has several advantages [Section 4.2] that make it stand out as well as a few limitations [Section 4.3] that need to be taken into consideration. Regarding future work, there are aspects presented in Section 4.4 that could be explored to possibly extend the method's utility and the system's effectiveness.

4.2 Advantages of the Method

As can be seen, the proposed method has several advantages:

- **It works with arbitrary tabular data** - the software does not need to know the columns of the table in advance. At runtime, the columns become properties of the data and the user configures the system based on them.
- **The configuration is declarative** - the configuration for the system requires the user to specify *what* he wants to see and the system adapts the user interface accordingly, as opposed to an imperative approach where the user would have to describe how to achieve the desired result.

- **The adaptive user interface is modular** - the modules that make up the presentation layer can be interchanged or swapped completely with new modules, which makes this method useful when facing different situations in which the interface might need specific elements.
- **The configuration does not depend on the actual data** - even if the data set is not yet known, provided that the system has access to its properties, the user can begin configuring his interface until the data becomes available.

4.3 Limitations of the Method

We also identified some limitations of the proposed method that need to be taken into account when using it:

- **The dataset cannot be filtered with complex conditions** - Using the current method, the dataset can be filtered only by assigning a fixed value for a certain property, disallowing more complex conditions.
- **Data needs to have a fixed structure** - Although this is a precondition given by the original issue, it is still worth mentioning as a limitation because data with fixed structure is only found in a fraction of the datasets from where humans need to draw conclusions.
- **The user cannot configure the source of the dataset** - Currently, the source of the dataset is static and the user has no control over it. The user can select a dataset that was uploaded but cannot derive datasets from different sources in a configurable manner.
- **Certain combinations of modules might be problematic** - Some combinations of user interface modules might end up being disorienting for a user. As an example, consider nested components that are scrollable in the same direction.
- **The method does not take into account any metadata** - No types of data are handled in a special manner by the system since all values are managed as a collection of characters. Numbers, colors and length measurements are examples of values that could be displayed differently to the user due to their meaning but are not.

4.4 Future Work

The proposed method is just a base idea that can be improved and the implemented system is only a proof of concept. Further study is required in order to bring the method to its full potential and therefore we present some aspects that could become the subject of this future research:

- **Valued properties filtered with complex conditions** - The values of a property could be filtered by comparing them to other values or by checking their inclusion in a given list of values.

- **Metadata analysis of the dataset** - The dataset could be configured with certain types of metadata, like the data type of the property or intended use. The adaptive interface and even the configuration interface could change based on this metadata to better fit the current use case.

- **Custom sorting orders** - The system should give the user the ability to sort the data in various ways that suit his needs instead of relying only on standard options.

- **Module suggestions** - The system could suggest what module can be used depending on some dataset metrics and previously chosen modules. For example, when a property has a large number of values available, the system could suggest a module that makes use of a drop-down menu for choosing the value, avoiding filling the user's screen with too much data at once. Another example would be to disadvise two consecutive modules that arrange data in the same direction in a scrollable manner to assure usability.

- **Performance improvements** - Several improvements can be made for the performance of the system. On the server's side, the data can be stored and handled in a different manner (not only with a graph database and the current structure) while still respecting the specification. Also, the dataset can be pre-classified according to some initial, more common configurations decided by the administrators. On the client device, the performance can be improved by fetching subtrees on demand from local memory for the modules that do not require all the data at once.

- **Interactive configuration** - The configuration of the user interface could be done via a more interactive experience, where the user drags and drops properties and modules, immediately previewing the result. This would make the process more intuitive for new users who do not have much experience with the process and might struggle to understand the notions of valued, classification and grouped properties.

Bibliography

[DC03] Frithjof Dau and Joachim Hereth Correia. Nested concept graphs: Applications for databases. *Submitted for ICCS*, 2003.

[DFAB04] Alan John Dix, Janet Finlay, Gregory D Abowd, and Russell Beale. *Human-computer interaction*. Pearson Education, 2004.

[Eli18] Roman Elizarov. Kotlin Coroutines, a deeper look. `https://elizarov.medium.com/kotlin-coroutines-a-deeper-look-180536305c3f`, 2018. Online; accessed 09 June 2021.

[Eli19] Roman Elizarov. Simple design of Kotlin Flow. `https://elizarov.medium.com/simple-design-of-kotlin-flow-4725e7398c4c`, 2019. Online; accessed 09 June 2021.

[EMK$^+$04] Andrew Eisenberg, Jim Melton, Krishna Kulkarni, Jan-Eike Michels, and Fred Zemke. SQL: 2003 has been published. *ACM SIGMoD Record*, 33(1):119–126, 2004.

[ES79] Dennis E. Egan and Barry J. Schwartz. Chunking in recall of symbolic drawings. *Memory & cognition*, 7(2):149–158, 1979.

[Frä21] Nicolas Fränkel. The state of JVM desktop frameworks: Jetpack Compose for Desktop. `https://nfrankel.medium.com/state-jvm-desktop-frameworks-jetpack-compose-db550f605e6a`, 2021. Online; accessed 09 June 2021.

[Gar11] Marko Gargenta. *Learning Android*. O'Reilly Media, Inc., 2011.

[GM95] James Gosling and Henry McGilton. The Java language environment. *Sun Microsystems Computer Company*, 2550:38, 1995.

[Hal11] Gary Hall. *Pro WPF and Silverlight MVVM: effective application development with Model-View-ViewModel*. Springer, 2011.

[Hed17] André Hedlund. How to Memorize and Retrieve Better Using the Memory Palace Technique. `https://edcrocks.com/2017/10/13/how-`

to-memorize-and-retrieve-better-using-the-memory-palace-technique/, 2017. Online; accessed 25 March 2021.

[Joh91] Tim Johns. Should you be persuaded: Two samples of data-driven learning materials. *English Language Research, University of Birmingham*, pages 1–16, 1991.

[Kan20] Siva Ganesh Kantamani. Jetpack DataStore: Improved Data-Storage System. https://betterprogramming.pub/jetpack-datastore-improved-data-storage-system-adec129b6e48, 2020. Online; accessed 09 June 2021.

[KJN06] Karl Kurbel, Anna Maria Jankowska, and Kamil Nowakowski. A mobile user interface for an ERP system. *Issues in Information Systems*, 7(2):146–151, 2006.

[LL07] Nelson KY Leung and Sim Kim Lau. No More "Keyword Search" or FAQ: Innovative Ontology and Agent Based Dynamic User Interface. *IAENG International Journal of Computer Science*, 33(1), 2007.

[Mor19] Caique Garutti Moreira. Building simple Webservices in Kotlin using Ktor. https://medium.com/playkids-tech-blog/building-simple-webservices-in-kotlin-using-ktor-59501f004070, 2019. Online; accessed 09 June 2021.

[O'L00] Daniel E O'Leary. *Enterprise resource planning systems: systems, life cycle, electronic commerce, and risk.* Cambridge university press, 2000.

[PEGM94] Angel R Puerta, Henrik Eriksson, John H Gennari, and Mark A Musen. Model-based automated generation of user interfaces. In *AAAI*, pages 471–477, 1994.

[Prz19] Baraniak Przemyslaw. UI Design in Practice: Gestalt Principles. https://uxmisfit.com/2019/04/23/ui-design-in-practice-gestalt-principles/, 2019. Online; accessed 25 March 2021.

[PS14] Candy Pang and Duane Szafron. Single Source of Truth (SSOT) for Service Oriented Architecture (SOA). In *International Conference on Service-Oriented Computing*, pages 575–589. Springer, 2014.

[Ric20] Leland Richardson. Understanding Jetpack Compose. https://medium.com/androiddevelopers/understanding-jetpack-compose-part-1-of-2-ca316fe39050, 2020. Online; accessed 09 June 2021.

[SB17] Stephen Samuel and Stefan Bocutiu. *Programming Kotlin*. Packt Publishing Ltd, 2017.

[SCG13] Joey Scarr, Andy Cockburn, and Carl Gutwin. Supporting and Exploiting Spatial Memory in User Interfaces. *Foundations and Trends® in Human–Computer Interaction*, 6(1):1–84, 2013.

[SMR16] Brenda Scholtz, Imran Mahmud, and T Ramayah. Does usability matter? An analysis of the impact of usability on technology acceptance in ERP settings. *Interdisciplinary Journal of Information, Knowledge, and Management*, 11(2016):309–330, 2016.

[Ste00] Constantine Stephanidis. *User interfaces for all: concepts, methods, and tools*. CRC Press, 2000.

[Str12] Bruno J. Strasser. Data-driven sciences: From wonder cabinets to electronic databases. *Studies in History and Philosophy of Science Part C: Studies in History and Philosophy of Biological and Biomedical Sciences*, 43(1):85–87, March 2012.

[SW09] Akash Singh and Janet Wesson. Improving the Usability of ERP Systems through the Application of Adaptive User Interfaces. In *ICEIS (4)*, pages 208–214, 2009.

[SWD$^+$20] Mike Stieff, Stephanie Werner, Dane DeSutter, Steve Franconeri, and Mary Hegarty. Visual chunking as a strategy for spatial thinking in STEM. *Cognitive Research: Principles and Implications*, 5(1), April 2020.

[TC99] David Thevenin and Joëlle Coutaz. Plasticity of User Interfaces: Framework and Research Agenda. In *Interact*, volume 99, pages 110–117, 1999.

[TSO19] Mirko Thalmann, Alessandra S. Souza, and Klaus Oberauer. How does chunking help working memory? *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 45(1):37–55, January 2019.

[Tur14] James Turnbull. *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.

[VWA$^+$15] Aleksa Vukotic, Nicki Watt, Tareq Abedrabbo, Dominic Fox, and Jonas Partner. *Neo4j in Action*, volume 22. Manning Shelter Island, 2015.

[War04] Colin Ware. *Information visualization: perception for design*. The Morgan Kaufmann series in interactive technologies. Elsevier, Amsterdam, second edition, 2004.