FH Vorarlberg
University of Applied Sciences

Application of Paradigms of HCI and Graphical Modelling to Improve Usability of Graphical UML Modelling Editors

Assessing changes to hierarchy, diagram composition, and navigation compared to an existing editor.

Master Thesis

Submitted in Fulfilment of the Degree
Master of Science.

Vorarlberg University of Applied Sciences
Computer Science MSc

Submitted to:
Dipl.-Ing. (FH) Walter Ritter

Handed in by:
Janik Mayr, BSc

Dornbirn, 10^{th} July 2022

Acknowledgements

First and foremost, I want to extend my gratitude to my supervisor Walter Ritter, from the department of User-Centred Technologies at the Vorarlberg University of Applied Sciences. They supported me with open questions, steered me to exciting topics related to my work, and gave excellent feedback in general. I want to thank you very much for your support.

Additionally, I would like to thank V-Research GmbH for enabling me to write this thesis as part of my work and for introducing me to UML/SysML modelling, from which the topic of the thesis originated.

Furthermore, I am grateful to my colleagues, friends, and fellow students for participating in my usability study and evaluating my thesis. In particular, I would like to thank Matthias Rupp for being a test person that I could ask quickly to get feedback during the creation and polishing of the implementation.

Lastly, I thank my parents for supporting me during my studies. They motivated me in times of COVID-19 lows and stressful times to push through.

Parts of the work were funded by the Austrian COMET Program (Project Tribology, no. 872176) and carried out at the "Excellence Centre of Tribology" (AC2T research GmbH) in collaboration with V-Research.

Abstract

The demand for managing data across multiple domains for product creation is steadily increasing. Model-Driven Systems Engineering (MDSE) is a solution for this problem. With MDSE, domain-specific data is formalized inside a model with a custom language, for example, the Unified Modelling Language (UML). These models can be created with custom editors, and specialized domains can be integrated with extensions to UML, e.g., the Systems Modeling Language (SysML). The most dominant editor in the open-source sector is Eclipse Papyrus SysML 1.6 (Papyrus), an editor to create SysML diagrams for MDSE.

In the pursuit of creating a model and diagrams, the editor does not support the user appropriately or even hinders them. Therefore, paradigms from the diagram modelling and Human Computer Interaction (HCI) domains, as well as perceptual and design theory, are applied to create an editor prototype from scratch. The changes fall into the categories of hierarchy, aid in the diagram composition, and navigation. The prototype is compared with Papyrus in a user test to determine if the changes have the effect of improving usability.

The study involved 10 participants with different knowledge levels of UML, ranging from beginners to experts. Each participant was tested on a navigation and modelling task in both the newly created editor, named Modelling Studio, and Papyrus. The study was evaluated through a questionnaire and analysis of the diagrams produced by the tasks.

The findings are that Modelling Studio's changes to the hierarchical elements improved their rating. Furthermore, aid for diagram composition could be reinforced by changes to the alignment helper tool and adjustments to the default arrow behaviour of a diagram. Lastly, model navigation adjustments improve a link's visibility and rating of a specialized link (best practice). The introduction of breadcrumbs had limited success in bettering navigation usability.

The prototype deployed a broad spectrum of changes that found improvement already, which can, however, be further improved and tested more thoroughly.

Kurzreferat

Der Bedarf bei dem Produkterstellungsprozess in der Verwaltung von Daten, über mehrere Domänen hinweg, nimmt stetig zu. Modell gesteuertes Systems Engineering (MDSE) ist eine Lösung für dieses Problem. Mit MDSE werden domänenspezifische Daten in einem Modell mit einer benutzenden-definierten Sprache formalisiert, z. B. mit der Unified Modelling Language (UML). Diese Modelle werden mit grafischen Editoren erstellt und spezialisierte Domänen können mit Erweiterungen des UML Standards, z. B. durch die SysML, integriert werden. Der grafische Editor im Open-Source-Bereich ist Eclipse Papyrus SysML 1.6 (Papyrus), ein Editor zur Erstellung von SysML-Diagrammen für MDSE. Bei der Erstellung eines Modells und dessen Diagramme unterstützt der Editor die Benutzenden nicht angemessen, oder behindert sie sogar. Daher werden aus den Bereichen Diagrammmodellierung und Human Computer Interaction (HCI) verwendet, sowie Wahrnehmungs- und Designtheorie angewandt, um einen Editorprototyp von Grund auf neu zu erstellen. Die Änderungen fallen in die Kategorien Hierarchie, Hilfe bei der Diagrammkomposition und Navigation. Der Prototyp wird in einem Benutzer- und Benutzerinnentest mit Papyrus verglichen. Dies stellt fest, ob die Änderungen eine Verbesserung der Benutzer- und Benuzerinnenfreundlichkeit erbringen. Die Studie umfasst 10 Teilnehmende mit unterschiedlichem Wissensstand über UML, von den Anfängern und Anfängerinnen bis zu den Expertinnen und Experten. Jeder Teilnehmende wurde mit einer Navigations- und Modellierungsaufgabe, jeweils im neu geschaffenen Editor namens Modelling Studio und Papyrus getestet. Die Studie wurde ausgewertet, mit einem Fragebogen und durch die Analyse der Diagramme, die bei den Aufgaben erstellt wurden. Das Ergebnis zeigt, dass die von Modelling Studio vorgenommenen Änderungen an den hierarchischen Elementen deren Bewertung verbessert haben. Außerdem konnte die Hilfe bei der Diagrammkomposition durch Änderungen an der Ausrichtungshilfe und Anpassungen am Standardverhalten der Pfeile eines Diagramms verstärkt werden. Schließlich verbessern die Anpassungen der Modellnavigation die Sichtbarkeit eines Links und die Bewertung eines spezialisierten Links (Best Practice). Die Einführung von Breadcrumbs hatte nur begrenzten Erfolg bei der Verbesserung der Benutzendenfreundlichkeit der Navigation. Der Prototyp enthält ein breiteres Spektrum von Änderungen, die bereits eine Verbesserung darstellten, die jedoch noch weiter optimiert und gründlicher getestet werden könnten.

Contents

List of Figures

List of Tables

List of Source Codes

List of Acronyms

API Application Programmable Interface

ATT Attractivenss

CAD Computer-aided Design

CC Class Diagram Criterion

DBSE Document-Based Systems Engineering

EMF Eclipse Modelling Framework

GC General Criterion

HCI Human Computer Interaction

HQ Hedonic Quality

HQ-I Hedonic Subquality Identity

HQ-S Hedonic Subquality Stimulation

i18n Internationalization

IDE Integrated Development Environment

ISO International Organization for Standardization

MDSE Model-Driven Systems Engineering

Papyrus Eclipse Papyrus SysML 1.6

PQ Pragmatic

REST Representational State Transfer

SysML Systems Modeling Language

TMM Tree Meta Model

UI User Interface

UML Unified Modelling Language

UX User Experience

V. Glossary

System

The International Organization for Standardization (ISO) defines in "ISO 9241-11:2018" that a system is the "combination of interacting elements organized to achieve one or more stated purposes" (International Organization for Standardization [ISO] 2018).

Usability

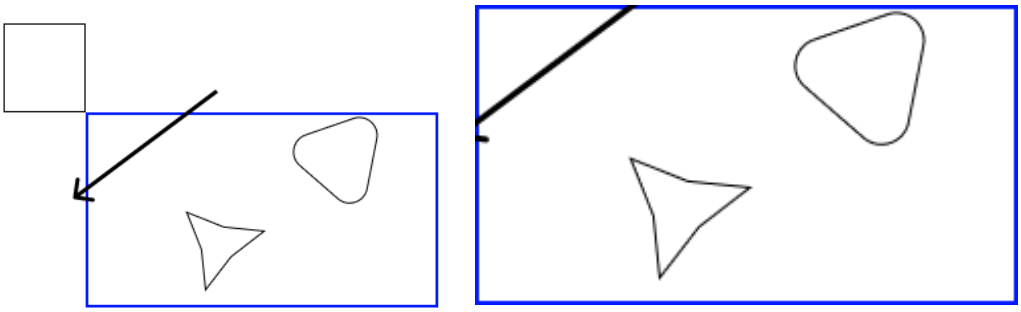
Based on "ISO 9241-11:2018", Usability is the "extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" (ISO, 2018).

User Experience

According to "ISO 9241-11:2018", User Experience (UX) is comprised of a "user's perceptions and responses that result from the use and/or anticipated use of a system, product or service". Responses and perceptions "include the users' emotions, beliefs, preferences, perceptions, comfort, behaviours, and accomplishments that occur before, during and after use". "User experience is a consequence of brand image, presentation, functionality, system performance, interactive behaviour, and assistive capabilities of a system [...]. It also results from the user's internal and physical state resulting from prior experiences, attitudes, skills, abilities and personality; and from the context of use." (ISO, 2018)

Viewport

A viewport is the area that is displayed after a projection transformation is applied. The displayed area is in the shape of a rectangle; the content can be a projection of a two- or three-dimensional scene. Figure V.1 shows an example of the content being bigger than the viewport available. In V.1a, the whole content is displayed with a blue rectangle indicating the current viewport, and the result of the projection transformation can be viewed in V.1b.



(a) Global View of Content with Viewport Marked in Blue

(b) Content of the Viewport as Seen on the Screen

Figure V.1.: Example of a Viewport (Source: Own Representation)

Accelerators

Aurora Harley writes in the Nielsen Norman Group's journal article of 2019 that accelerators are "[a]lternate methods for accomplishing a frequent action in a user interface [to] support expert users by speeding up their interactions, without hindering novices." These methods include macros and most importantly keyboard shortcuts. (Aurora Harley 2019)

Context Menu

Anna Kaley summarizes Context Menus on the Nielsen Norman Group's journal article of 2019 as follows: "Contextual menus are displayed on demand and contain a small set of relevant actions, related to a control, a piece of content, a view in an app, or an area of the UI. When designed right, they deliver relevant tools for completing tasks without adding clutter to the interface." (Anna Kaley 2019)

1. Introduction

With the growing complexity of creating a product, the effort to keep the overview and assure correctness increases. The previous method to create a product was described as Document-Based Systems Engineering (DBSE) and had the different parts of the product documented in separate documents. One disadvantage of this method is that people could work with different document versions. Additionally, separate documents make it difficult to find or require spending extra time looking for the relevant information. Furthermore, for information from different teams, a specialized program might be needed, for which the individual does not have a licence or requires special knowledge about how to get the relevant information.
Instead of creating static documentation like in Document-Based Systems Engineering (DBSE), with the new method of Model-Driven Systems Engineering (MDSE), a dynamic model is necessitated from which the static documentation can be created. For this dynamic model, the Unified Modelling Language (UML) standard is used; for applications in engineering domains, the Systems Modeling Language (SysML) expands the standard. This method is gaining traction and finds more adopters. (Bajaj et al. 2016)
During the process of engineering a product following MDSE, different domains merge to create said product. A selection of the given domains is product management, requirements engineering, simulation, Computer-aided Design (CAD), optimization and manufacturing. The different domains use specialized tools to achieve their tasks, making knowledge sharing harder because the information is formalized in the tools' format and may not be understandable by other teams. SysML is the formalization framework to bridge the gap between the domains and make the whole system more understandable to the other teams. Knight and Munro state that the discipline of "[s]oftware visualisation [] makes use of various forms of imagery to provide insight and understanding and to reduce complexity of the existing software system under consideration" (Knight & Munro 1999: p.3). A follow-up work by Tilley and Huang checked the viability of UML diagrams for their usefulness as a software visualization and found them to be "a convenient mechanism for software engineers to represent high-level system designs" (Tilley & Huang 2003: p.188).
MDSE with SysML also has the advantage that, for example, engineers can reuse existing components in a new product and import them as part of the SysML model. In this example's case, a metal screw is needed. The requirements engineers can look into the model if one single screw for the requirements already exists; if multiple, the optimization domain can optimize the screw for, for example, its cost, strength, and production complexity. Assuming it does find a screw, it is imported into the SysML diagram for the product part. Now other departments know which screw to use. From the SysML, the CAD department knows the CAD model to import and where it is stored. Based on multiplicities in the SysML diagram, the manufacturing department can also calculate the number of screws needed for the product for their bill of

materials.
In general, two types of editors emerged for UML, programming and visual editors. In programming editors, the layout of the elements is programmed and then rendered. Visual editors do not make that differentiation and just allow the user to model the diagrams visually and move elements around freely. Focusing on visual editors for UML and SysML in the paid sector are IMB's Rhapsody and MagicDraw. Offered by the open-source sector is Eclipse Papyrus, which supports UML and SysML (extra plugin, also free).

1.1. Motivation

As part of research activities in the domain of constraint solving and optimization for models in MDSE usability frictions became apparent. While working on the model and diagrams, usability frictions arose with the modelling editor itself. For these activities, the chosen editor is Eclipse Papyrus SysML 1.6 (Papyrus) due to easy extendability for the planned features via Eclipse Plugins and the open-source nature of Papyrus itself. The UML standard does not include much in the readability of diagrams (Eichelberger 2003). Thus, it is up to the tools to design the interactions on how to create or edit the diagrams.
With the increased usage of Papyrus, it was noticed that interacting with the model required much scrolling in the property view if the diagram area should fit a bigger diagram. Furthermore, some interactions are strange to get used to for inexperienced individuals. Others who used the program for a longer time resorted to using a key bind for the specific action or diving into the settings to make adjustments or adding interactions to an Eclipse Plugin in code. Lastly, the created diagrams "do not look good" if a modeller does not pay attention to aesthetics or tries to go fast.
Addressing these problems and resolving inconveniences was the motivation behind creating a new editor prototype. This editor should be built from the ground up based on findings from literature to improve the experience in modelling and viewing.

1.2. Goals

The thesis's first goal is to analyse the reference editor, Papyrus, in terms of usability. Secondly, the discovered shortcomings are converted into changes that should improve the usability based on state-of-the-art recommendations, such as laws and best practices. The third goal is to develop a prototype editor based on the suggested improvements. The editor should provide users with an interface with which a diagram can be modelled. Diagrams should be able to display the subset of elements for Block-Definition, and Package Diagrams from the SysML standard. Finally, the prototype is evaluated against Papyrus in a user test to assess their usability. The aim is to improve the usability through the changed elements and maintaining at least the same usability in unchanged parts.

1.3. Thesis Overview

The thesis is organized in the steps taken during the prototype development. Chapter 2 introduces state of the art for modelling and usability paradigms and theoretical frameworks of perceptual theory. Based on state of the art, the editor Papyrus and a diagram created with it is analysed in chapter 3. The problem definition analysis identifies weak points that could be improved upon, some of which are incorporated into the concept. Chapter 4, concept, proposes improvements to identified weak points based on state of the art. The implementation of the prototype is detailed in chapter 5. It explores the implementation of elements named in the concept and general components, as well as the technologies and libraries that are used. The evaluation of the prototype is detailed in chapter 6. It introduces the methodology, results of the user test, and the context of results compared to state of the art. Finally, chapter 7 discusses the results in the context of the implementation and closes with an outlook for further research.

2. State of the Art

2.1. Perceptual theory

Saji defines perceptual theory as a "paradigm that sensory information processing in human cognition, such as perception, recognition, memory, and comprehension, are organized and shaped by our previous experience, expectations, as well as meaningful context" based on prior work by Solso (1998) (Saji 2014).
The articles "On Understanding Software Tool Adoption Using Perceptual Theories" (2004) and "On evaluating the layout of UML diagrams for program comprehension" (2006) by the authors Wong and Sun summarize literature related to perceptual theory targeted at software tools and UML diagrams.
Wong and Sun elaborate that "[t]he laws of perception explain how our visual system identifies objects and how we put together basic features to observe a coherent, organized world of things and surfaces. From a software visualization perspective, the principles of perceptual organization provide the basic design rules to organize multiple artifacts, so that users can group related information and segregate useful information easily and without ambiguity" (Wong & Sun 2006: p.234).

2.1.1. Theories of perception

Visualization is the basis on how people understand information, but the processing of stimuli in the brain is still unclear (Wong & Sun 2006). Petre, Blackwell, and Green list in the paper "Cognitive Questions in Software Visualisation" (1996) the following theories:

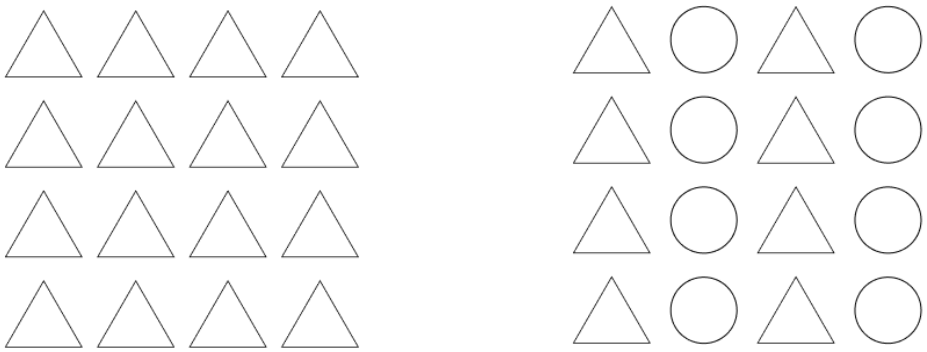
- **Gibson's theory**: People build a cognitive map of how to interact with the outside world by adjusting their attention to the physical features of their surroundings. (Gibson 1979)
- **Marr's theory**: Mental constructs, considered to be cognitive functions, act as filters for transforming raw visual stimuli into information. (Marr 1982)
- **Gestalt theory**: This theory explains why some representations are better than others by formulating the principles of organization. It is based on the human perception, which has been restructured to make it more unified and coherent. (Benjafield 1992; Moore & Fitz 1993)
- **Theory of notation**: Effective notations for visualizations, including symbol systems for graphs, to aid in the process of explaining semantics. (Petre et al. 1996)

2.1.2. Perceptual organization

Perceptual organization describes how the human eye processes information, in terms of grouping of elements to create larger collections. (Goldstein 2010) Perceptual organization can be defined by the following six rules:

- **Law of good figure (Prägnanz):**
  Following the meaning of Prägnanz from German of simplicity, regularity, stability, and conformity the Law is also called "Law of Simplicity" or as a combination of the attributes "Law of good figure". Goldstein stats that "[e]very stimulus pattern is seen in such a way that the resulting structure is as simple as possible" (Goldstein 2010).

- **Law of similarity:**
  Grouping of elements if influenced by the similarity of the elements, for example by their shape, colour, size, and rotation. In figure 2.1 the effect is visualized. In 2.1a the layout is not determined because the elements are the same, but in 2.1b the elements form columns as they are different in shape. (Wong & Sun 2006)



(a) (b)

Figure 2.1.: Examples of the Law of Similarity, Adopted from Wong and Sun (2006: p.236, fig.1)

- **Law of continuation:**
  Points that form a line or smooth curve are more likely to be observed as a group. Resulting and existing lines are combined to create the smoothest path. (Wong & Sun 2006) This can be observed in figure 2.2.

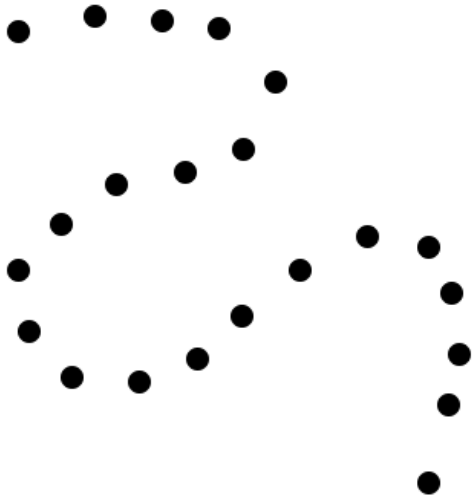


Figure 2.2.: Examples of the Law of Continuation (Source: Own Representation)

- **Law of proximity**
  The distance between objects determines if they are perceived as a unit. With small enough distances, the law of similarity can be overpowered. Figure 2.3 shows the same example of objects as the law of similarity, but rows are perceived instead of columns.

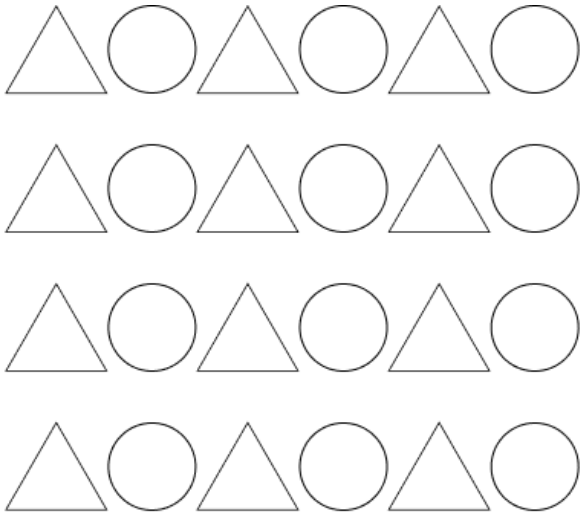


Figure 2.3.: Examples of the Law of Proximity. Adopted from Wong and Sun (2006: p.236, fig.2)

- **Law of connectedness:**
  Visually connected elements show higher likelihood to be grouped as a unit. Figure 2.4 demonstrates that the binding of two dots is stronger than the Law of Proximity. The law of proximity should apply here because the dots that share a connection are further apart than the ones that do not. (Wong & Sun 2006)

Figure 2.4.: Examples of the Law of Connectedness. Adopted from Wong and Sun (2006: p.236, fig.3)

- **Law of familiarity**
  As the name of the law suggest, if figures are familiar or meaningful, they have a higher chance of being grouped together. (Wong & Sun 2006)

2.1.3. Perceptual segregation

Perceptual segregation describes how an object, also called figure, is separate from the ground (background) it is on. (Goldstein 2010)

- **Law of symmetry**
  Symmetric features are seen as a distinct figure. (Goldstein 2010)
- **Law of orientation**
  Elements oriented vertically or horizontally are seen as a figure over other rotations. (Goldstein 2010) Figure 2.5 shows that for this figure, it is more likely to see the plus instead of the cross.

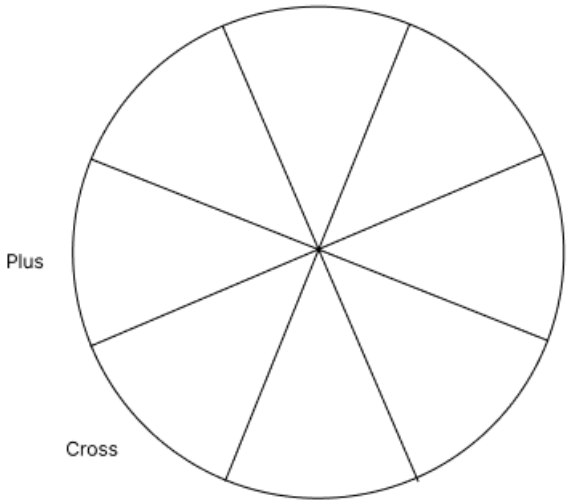


Figure 2.5.: Examples of the Law of Orientation. Adopted from Wong and Sun (2006: p.237, fig.4)

- **Law of contour**
  Border ownership is a property of figures that creates a distinct separation from figure and background. (Goldstein 2010)

2.2. General Design Principles

Design principles are a collection of biases, considerations, guidelines, heuristics, and laws that can be applied to design to have good design and usability.

2.2.1. Nielsen's 10 usability heuristics

In 1994, Jakob Nielsen developed 10 usability heuristics which are based on earlier work with Rolf Molich in 1990 (Nielsen & Molich 1990). The so-called "Nielsen's 10 usability heuristics" are "a "discount usability engineering" method for evaluating user interfaces to find their usability problems" (Nielsen 1994: p.152).
These heuristics or named factors by Nielsen that affect the perception of a system or application are:

Visibility of system status

The system should "keep [the] user informed about what goes on" by "provid[ing] status information", "feedback [] for all actions" taken by the user and "indicate progress in task performance" in a "timely and accurate" manner (Nielsen 1994: p.153).

Match between system and the real world

The system should "speak the user's language" and convey it in "familiar terms and natural language", should make use of "familiar user's conceptual model" and follow "real world conventions" (Nielsen 1994: p.153).

User control and freedom

The top four heuristics of this factor relate to the forgiveness of a system. The main point is to "make actions reversible" this can be achieved if "undo and redo [is] supported" and is achieved by following an "obvious way to undo actions" (Nielsen 1994: p.153). Regarding closing pop-ups or cancelling an action, the system should make sure "Clearly marked exits" exist. This is especially important if a user opened a dialogue or started a task mistakenly (Nielsen 1994: p.153).

Consistency and standards

The system should be consistent in "expess[ing] [the] same thing [the] same way", making the "same thing look the same", "conform to platform interface conventions" and offer "consisted key definitions throughout" (Nielsen 1994: p.153). For example, a button with the task to cancel the current objective's dialogue should always be consistently labelled either "cancel" or "abort", the placement should follow the platform, most likely the bottom right and look like a button. Furthermore, the dialogue should also be cancellable by pressing escape (Nielsen 1994: p.153).

Error prevention

The system should "prevent errors from occurring in the first place" by "design[ing] [it] to prevent errors". A simple example is to disable buttons before the user interacts with them and receives an error message that inertial conditions are not meet. Also tell the user if an action would result in an inconsistent state of the application or loss of data, e.g., remember the user to save on close (Nielsen 1994: p.153).

Recognition rather than recall

The system should display all possible actions and options in the User Interface (UI) that can be taken in a "see-and-point [style] instead of remember-and-type". To further "minimize the users' memory load" the options should be sorted in lists. The addition of "icons and other visual indicators" can also help to provide the user with already familiar actions from other programs, e.g., the floppy disk as the save icon. The result of an action should also be visible to avoid the user questioning if something has happened (Nielsen 1994: p.153).

Flexibility and efficiency of use

To increase the flexibility and efficiency of use, the system should provide accelerators (see. V). In addition, the "[s]ystem should be efficient to use" (Nielsen 1994: p.153) and "Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users" (Nielsen 1994: p.156).

Aesthetic and minimalist design

Jakob Nielsen claims in the updated version of 2020 that "[i]nterfaces should not contain information which is irrelevant or rarely needed. Every extra unit of information in an interface competes with the relevant units of information and diminishes their relative visibility" (Jakob Nielsen 2020).

Help users recognize, diagnose, and recover from errors

Users should be able to recover from errors that occur in a program. To accomplish that, errors must be visible when they happen. For a given error message, Jakob Nielsen defines that they "should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution" (Jakob Nielsen 2020).

Help and documentation

In an ideal world, a system should be understood and usable without documentation, but if it is not, Jakob Nielsen defined heuristics for help and documentation. Following the article from 2020 documentation should be "easy to search", "focused on the user's task", "concise", and provide a "list [of] concrete steps that need to be carried out" Jakob Nielsen (2020).

2.2.2. Fitts's Law

Fitts's law describes the time it takes a person to move to and click a target. The relation can be expressed as shown in equation 2.1. ID describes the difficulty to point to a target, equation 2.2 reveals that the information is in bits taken from the distance D to the element and its width W. (Fitts 1954)

$$T = a + b * ID \tag{2.1}$$

$$ID = \frac{2D}{W} + 1 \tag{2.2}$$

$$T = a + b * log_2(\frac{2D}{W} + 1) \tag{2.3}$$

where

- T: Average Time to complete the movement
- a & b are constants that can be measured and are different for the used pointer device
- D: Distance from starting point (also called Amplitude)
- W: Width along axis of movement

The mathematical connections are illustrated in figure 2.6.

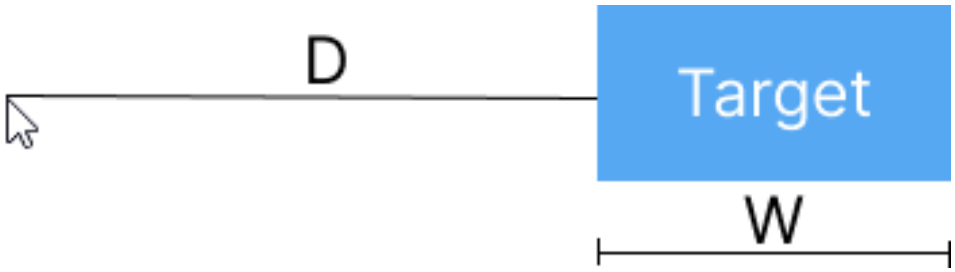


Figure 2.6.: Fitts's Law for Movement Along Horizontal Axis (Source: Own Representation)

2.2.2.1. Difference in Physical to Virtual Pointing

Graham and MacKenzie studied the difference between using physical and virtual pointing devices and proposed the theory that "movement planning is similar for both virtual and physical pointing. The difference between the virtual and physical display is apparent only in the second movement phase, where visual control of deceleration to the smaller targets in the virtual task took more time than in the physical task." (Graham & MacKenzie 1996: p.297)
Figure 2.7 shows the phases for the horizontal example.

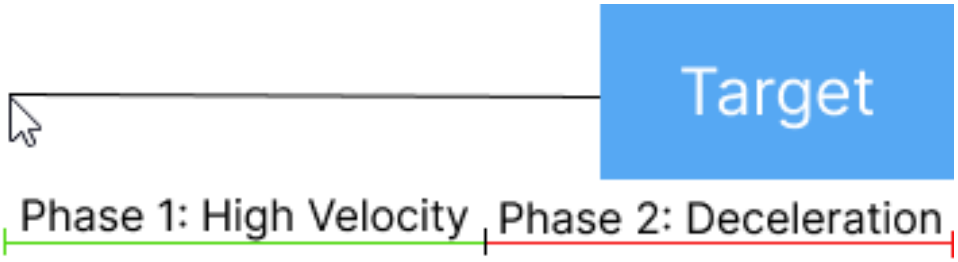


Figure 2.7.: Fitts's Law Movement Phases for Movement Along Horizontal Axis (Source: Own Representation)

2.2.2.2. Rule of Infinite Edge

Fitts's law has an additional rule that only applies to the edges of screens. A mouse pointer is stopped at the edges of the display, even if it moves further in the physical space. (Hale 2007) In practice, this creates a target with an infinite width along the axis of movement, figure 2.8.

Figure 2.8.: Fitts's Law Rule of Infinite Edge for Movement Along Horizontal Axis (Source: Own Representation)

The infinite width can decrease the second phase of deceleration to increase accuracy. For example, on Windows, the Windows Button is located at two infinite edges, the left and the bottom side of the monitor.
Combining the different aspects of Fitts's law a button should be as close as possible to the mouse pointer for best times, e.g., Context Menu (see V) or if it is further away, placement alongside an edge increases the width of the target.

2.2.3. 8pt Grid Rule/Guide

For the 8pt grid, multiples of eight are used to define an element's dimensions and padding. Applying this grid system results in a consisted looking UI. Furthermore, the sizing in multiples of eight provides better fractions on different screen sizes compared to multiples of ten. Table 2.1 shows the most common screen resolutions from June 2021 to June 2022 according to StatCounter (2022) and if they are multiples of 8 or 10.

Resolution	Marked Share June 21 - 22	Multiples of 8		Multiples of 10	
		Width	Height	Width	Height
1920 x 1080	22.05%	✓	✓	✓	✓
1366 x 768	18.92%	X	✓	X	X
1536 x 864	10.16%	✓	✓	X	X
1440 x 900	6.18%	✓	X	✓	✓
1280 x 720	5.83%	✓	✓	✓	✓

Table 2.1.: 8pt and 10pt UI grid system compatibility applied to 5 most common screen resolutions from June 2021 to June 2022. (Source: Own Representation, Data: StatCounter (2022))

In general, screens have even size dimensions, and thus using an uneven spacing could quickly cause problems. Common sizes for the 10pt system include at a factor of 0.5 -> 5pt, 1 -> 10pt, 1.5 -> 15pt If you want to split an element that has an uneven number for a size the content has halves of pixels to deal with. In the 8pt system, however, all splits can be split further as after a split the size still is even (e.g., 32 -> 16 -> 8).

2.3. Modelling Related Principles

2.3.1. UML and SysML terminology

The following is a selected subset of UML and SysML terminology that is used in this thesis.

UML or SysML element

A UML or SysML element is an element defined by the relevant standard. For example, a Block is an element from the SysML standard. However, all SysML elements are based on UML elements, and thus all SysML elements are also UML elements.

Model

A model is the root organizational unit for a UML project, its internal structure is a tree that contains nested levels of UML elements and diagrams. In practice, the model is split into two trees that are merged inside the editors, the first one being the actual UML elements and secondly, diagrams, links, and non-UML elements that are in diagrams. If the first part follows the ECORE format, other editors can read this model as well.

Diagram

A diagram represents a part of the model visually. Most commonly, a diagram shows the elements of the current subtree in the model. Diagrams are used to show the connections between elements and highlight specific values of elements.
Different diagram types are better suited to specific tasks, the two diagram types addressed in this thesis are: Firstly, the package diagram is useful for getting an overview of packages in its subtree and for navigating to a package by interacting with it. Secondly, the class (UML) or block

definition (SysML) diagram showcases the most important properties and their connections are visualized with arrows.

2.3.2. Hierarchical Editor Parts

The editors used for visual modelling share a common set of hierarchical elements that exist in the editors. These elements are also called Views or Parts and are listed below.

Menu bar

The menu bar holds interactions like the "File", "Edit" and other menus that are commonly found in applications on computers.
The menu bar is commonly merged with the application top bar that integrates the minimize, maximize, and close buttons.

Tool bar

The toolbar is a part of the editor that offers shortcuts to commonly used interactions for the editor.

File Tree / Model Explorer

The File Tree or Model Explorer is a view that displays the contents of the model. The way the model is displayed is called a tree view because it follows the tree like structure of the model, which is composed of different levels of packages, blocks, properties, etc.

Diagram Area

The diagram area is the part holding the actual SysML diagram. This part allows the interaction with the model, like moving, editing, deleting, partly creating of SysML elements.

Tabbar

The tabbar allows the user to switch diagrams and close them.

Palette

The palette is used to create new SysML elements inside a diagram. To create a new SysML element of a desired type, two main interactions exist. Both interactions start by searching for the element in the palette. It can then be moved in the diagram by drag and drop, or the user can select it in the palette and click in the diagram.

Property View

The property view or properties view shows the complete information about an object that is selected either in the model or the diagram. The contents of the property view differs for the selected element based on the type of the SysML element. Furthermore, the property view allows the user to change all editable values of the selected SysML element.

2.3.3. Criteria for diagram comprehension

Wong and Sun propose in their paper "On evaluating the layout of UML diagrams for program comprehension" criteria and guidelines that increase the comprehension of sequence and class diagrams. For this thesis, the subset of General Criterion (GC) and Class Diagram Criterion (CC) is relevant. Although the study focused on automatic layout for diagrams, the findings are also relevant for diagrams created by humans, as the purpose and target group stays the same. The established criteria assess the readability of generated diagrams, but can be applied to all diagrams, as they also should be read and understood by humans. (Wong & Sun 2006)
In figure 2.9, the general and class diagram criteria are sorted into the laws of perceptual organization (2.1.2) and perceptual segregation (2.1.3).

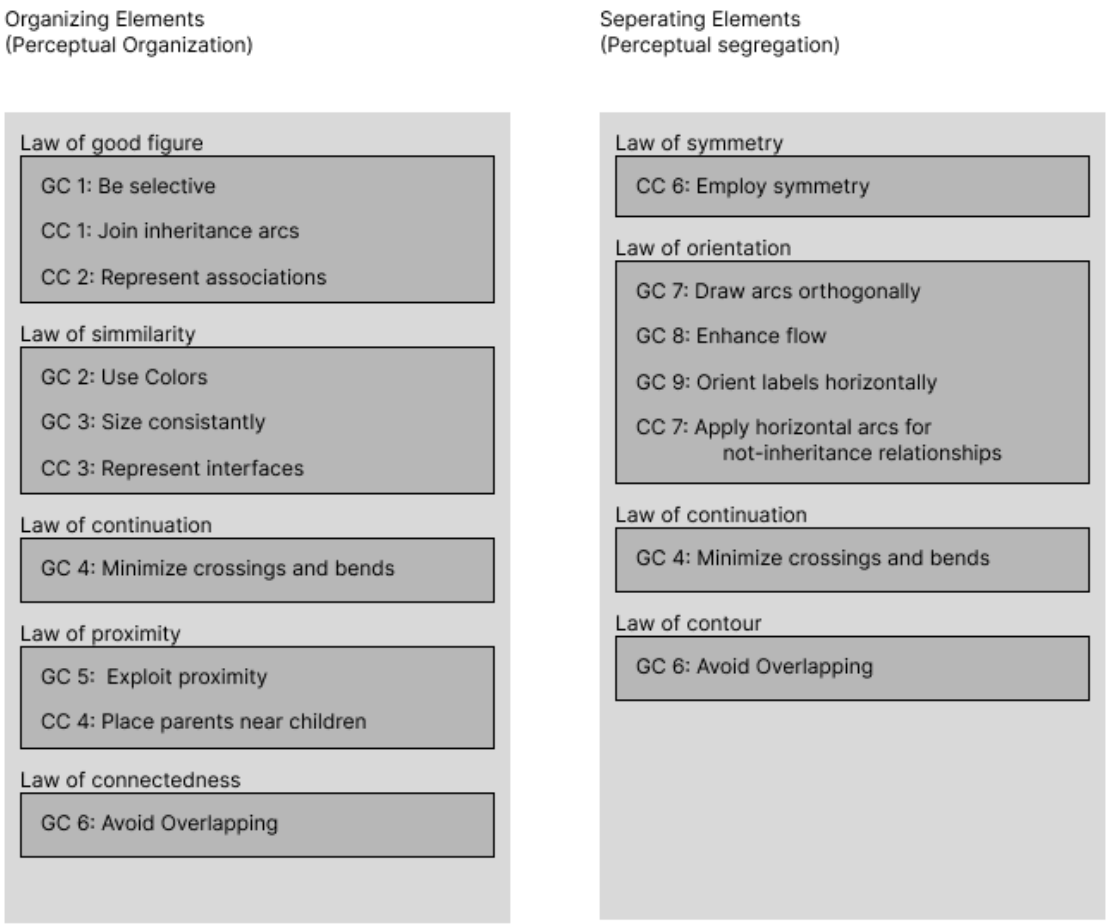


Figure 2.9.: UML Diagram Criteria Categorized by Perceptual Organization and Segregation Laws. Adopted from (Wong & Sun 2006: p.238, fig.5, removed not applicable rules)

2.3.4. Comparison Papyrus vs Magic Draw Paper

Planas and Cabot conducted a user test comparing Papyrus and MagicDraw in their paper "How are UML Class Diagrams built in practice? A usability study of two UML tools: MagicDraw and

Papyrus" 2020. The user test had 45 participants of the target group, students. The task for the participants include creating a project, diagram and performing create, read, update, and delete (CRUD) interactions on a diagram to create the same diagram they were provided with. Planas and Cabot analyse three research questions connected to UML modelling and usage of editors. The first research question, concerned the strategy users follow to create a diagram. In 93% of the tests, the users started with a Class, but no significant continuation strategy was found. However, a user who uses a depth strategy (class shapes with properties first) finished the task quicker and with fewer clicks on average (9 min, 133 clicks) compared to a user following the breadth approach (class shapes are connected to other elements before properties) (10 min, 127 clicks). The slowest users used a mixed approach between both strategies (11 min, 133 clicks). Planas and Cabot state that an opinionated mode of the editor to force a depth strategy could help beginners that are overwhelmed with the freedom of the editor.
Research question two addressed the speed of modelling with which the different strategies operate. They found that the overall effort and efficiency could be improved for both editors, but drawbacks are associated with obstacles created by the editors (Papyrus more than MagicDraw) and freedom given to the modeller, which hinders users without program knowledge.
The last research question, number 3 highlights common obstacles during the modelling progress. Obstacles are missing documentation, editors require a not obvious order of creation steps (not needed from standard perspective), too many menu points (they stated that users had to start over in searching menus and context menus for commands to finish their task).

3. Problem definition

This chapter will address problems encountered when modelling in editors, particularly Papyrus. The origin of the usability problems is from not exemplary implementations and missing features. The thesis addresses three main problems, which are discussed in detail below. At first, a general problem is introduced, 3.1, afterwards the first problem with the hierarchy in Papyrus is addressed in 3.2. Followed by missing aid for good diagram composition, 3.4 and lastly, 3.6 closes with improvements to the navigation through diagrams.

3.1. General

Generally said, the application does not feel intuitive to use. Following Raskin's definition of intuitive in terms of HCI from the word intuit describes the process of understanding a concept effortless or without prior exposure to it (Raskin 1994: p.17). In terms of Papyrus, understanding the first few screens is a considerable effort, because of the inclusion of many elements in the toolbars, plethora of views that can be opened, and general overload of things that are available. The last point not directly the fault of Papyrus as it is a Plugin for an Integrated Development Environment (IDE), Eclipse, and many icons, buttons, and views are relics from the base editor. Starting to model, a simple task of creating a property can result in problems not understanding which element to use from the palette, as multiple different with "Property" in their name exist.

3.2. Hierarchy

The hierarchy of an editor can be broken down into functional parts. For a modelling editor, these parts are the diagram area, file tree/model explorer, properties view, palette, tab bar, toolbar, and menu bar. More details on each of the parts can be found in section 2.3.2 on page 23. An example for the Papyrus Editor can be seen in figure 3.1 and the figure 3.2 provides a simplified overview for the part names.

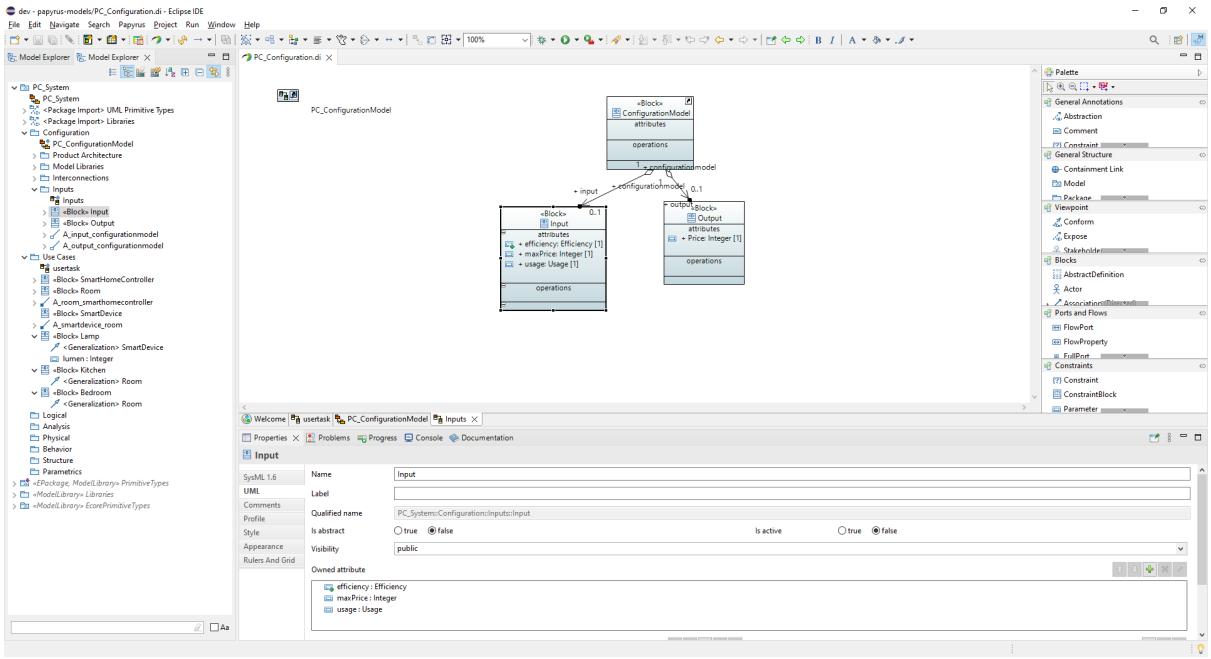


Figure 3.1.: Example of Papyrus with Open Diagram (Source: Own Representation)

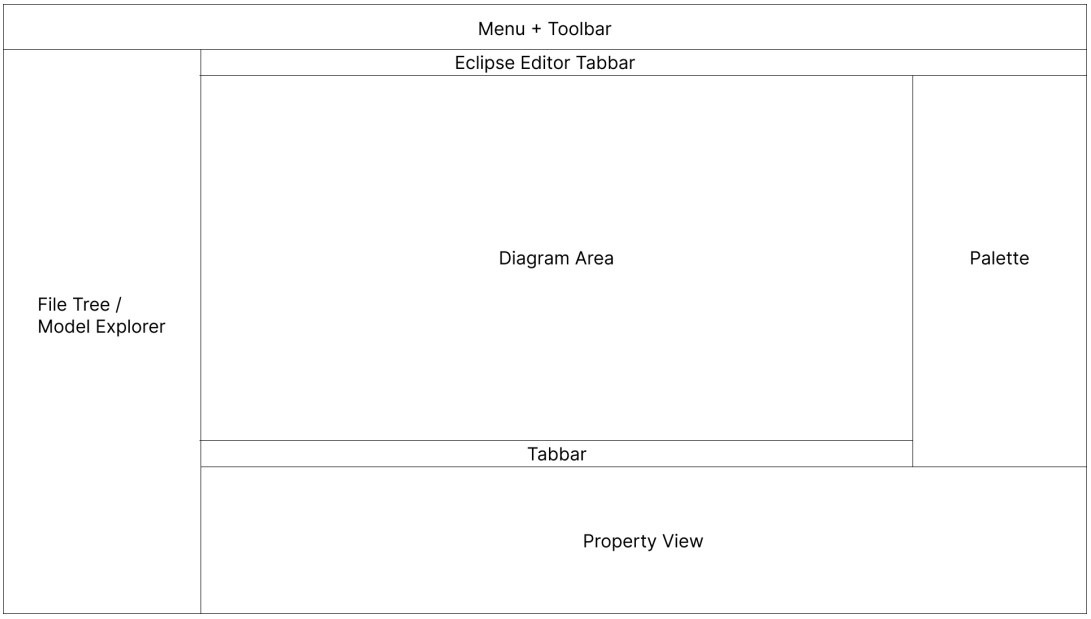


Figure 3.2.: Papyrus Simplified Representation of Editor Parts (Source: Own Representation)

3.2.1. Hierarchical Parts

3.2.1.1. Diagram Area

Generally speaking, the diagram area should provide a large area to feature the diagram content itself. The diagram is the application's content, and its visualization is how the information is shared between users. Therefore, adequate space must be provided for the component.

3.2.1.2. File Tree / Model Explorer

The file tree is an essential part of the application that represents the primary way of navigating through the model contents. In Papyrus, the user can also navigate through hyperlinks in the model itself, but these need to be created manually.

3.2.1.3. Property View

The 'property view' or 'properties view' is used to view the properties of elements in the model. It provides additional information not directly shown in the model and provides inputs to change the values for an element. The property view at the bottom of the screen is not ideally positioned because the most common field that the user wants to edit is the element's name. However, this input element is not close to any edge; thus not making use of the law of infinite edges. Additionally, the properties view often requires resizing or scrolling to reveal the other properties that the user wants to view.

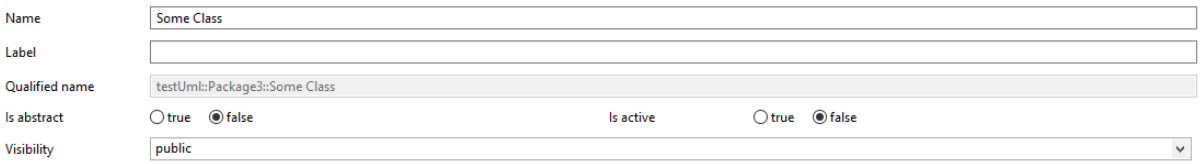


Figure 3.3.: Papyrus' Property View (Source: Own Representation)

Figure 3.3 shows that the horizontal space is rarely used to its full potential, the inputs just grow with the screen size. The growing inputs produce another problem of increased distances for the eyes and the mouse to travel to achieve tasks. In the figure, 3.3, above the last element "Visibility" is a drop-down that is opened by clicking on the chevron button on the right. The large distance decreases the usage of the button, leading the user to press into the text box and start typing instead. The only UML element that fills most of the space are "member ends" of an association. In addition, each row in the properties view can feature multiple inputs that are not necessarily logically related, increasing the effort to find a specific property. The increased efforts are connected to disregarding the theory of perceptual organization (law of similarity, proximity, continuation).

3.2.1.4. Palette

The palette is used for creating a new element inside a diagram. An example for the element can be seen in figure 3.4. The content of the palette is organized in an accordion style. To see all items, the user has to expand or minimize the different categories (in figure as 1 annotated) that include the items. Additionally, the palette requires the user to click a button to "scroll" up (in figure as 2 annotated) and down inside a category if it does not fit on a page.

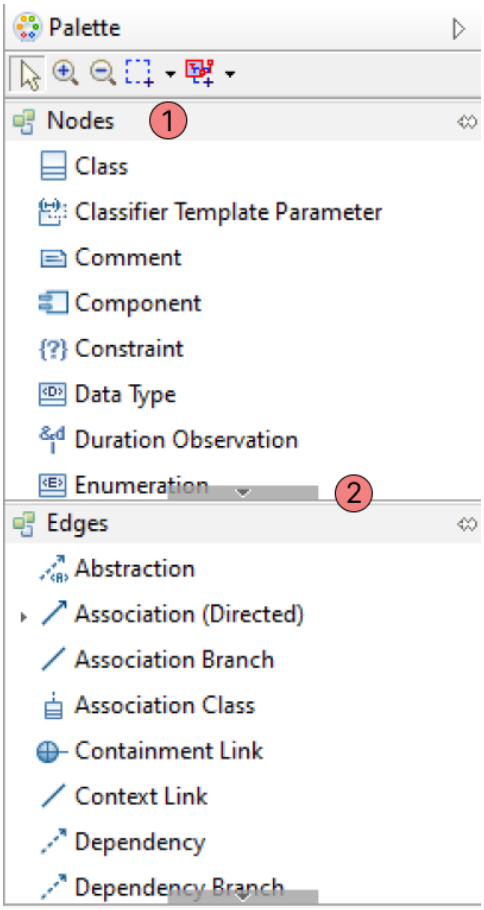


Figure 3.4.: Example of Palette in Papyrus (Source: Own Representation)

This disregards the heuristic, Recognition rather than recall, by Nielsen and Molich (1990). Additionally, the effort to search specific elements in the palette increases.

3.2.1.5. Tab-bar

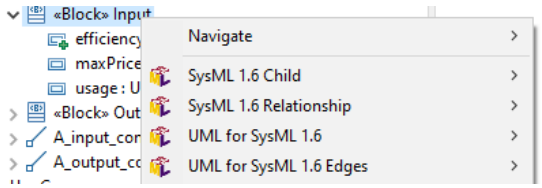
The tab-bar is positioned below the diagram and above the 'properties view'. The usual position of the tab-bar is at the top of the diagram area, where Eclipse has its Editor tab-bar. The duplication of the tab bar is unnecessary from the user's perspective and only creates confusion about which of them is the correct one. Both Google (2022b); Jakob Nielsen (2016) recommend the placement of the tab-bar above the content, which can be changed by it.

3.3. Potential Hierarchical Optimization

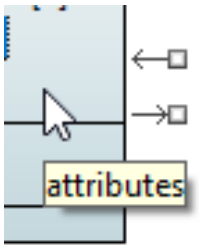
In the case of Papyrus, the editor windows are not optimized for their importance to the modelling process. The 'property view' is used more often than the palette during viewing and editing for the modelling process. In addition, alternative processes exist to create a new element, which is the single task of the palette. The alternate processes for creating a new element in the model are:

1. Context Menu started from the explorer or diagram area itself (e.g., Fig. 3.5a)

2. Accelerators for creating a specific element in the current diagram
3. For arrows, hover drag-points from existing elements in the diagram can be used (e.g., Fig. 3.5b)



(a) Context Menu from Model Explorer



(b) Create Arrow from dragging on in-diagram controls

Figure 3.5.: Alternative Element Creation Processes (Source: Own Representation)

Following Fitts's Law and the concept of infinite edges, the positioning along an edge is better for buttons that have to be interacted with often and cannot be placed close to the mouse cursor's most likely spot. In this case, the regular place of the cursor is the diagram area or the model tree.
Papyrus allows the user to move the view to custom positions, but this does not resolve the issues because the view's content does not adjust to the new dimensions of the view. For example, if the 'properties view' is moved to the right side, the content does not change. The user still has to expand the view to see the options. Additionally, the input fields would have concealed action buttons on their right side.
The emerging research question is: Can the hierarchy of the editor be changed to follow the theories of perceptual organization and perceptual segregation, as well as optimize the mouse movement following Fitt's law for the priority given by the usage frequency of an editor part?

3.4. Aiding Good Diagram Composition

A user modelling a diagram with limited support by the editor results in inconsistency. This inconsistency exists in the following forms:

1. Interpersonal differences
2. Positioning of elements
3. Alignment of elements
4. Spacing of elements
5. Sizing of elements

The forms of inconsistency can be observed in diagrams that is used as a basis for analysing and testing. A co-worker created this model as a use-case example for MDSE related tasks in Papyrus. The creator is a mechanical engineer by profession with an intermediate knowledge

level of modelling. From the modeller's perspective, the diagrams are considered "done" and are "production ready". A collection of problematic examples can be seen in figures 3.6, 3.7, and 3.8; however, most other diagrams of the model had similar issues as the ones raised below.

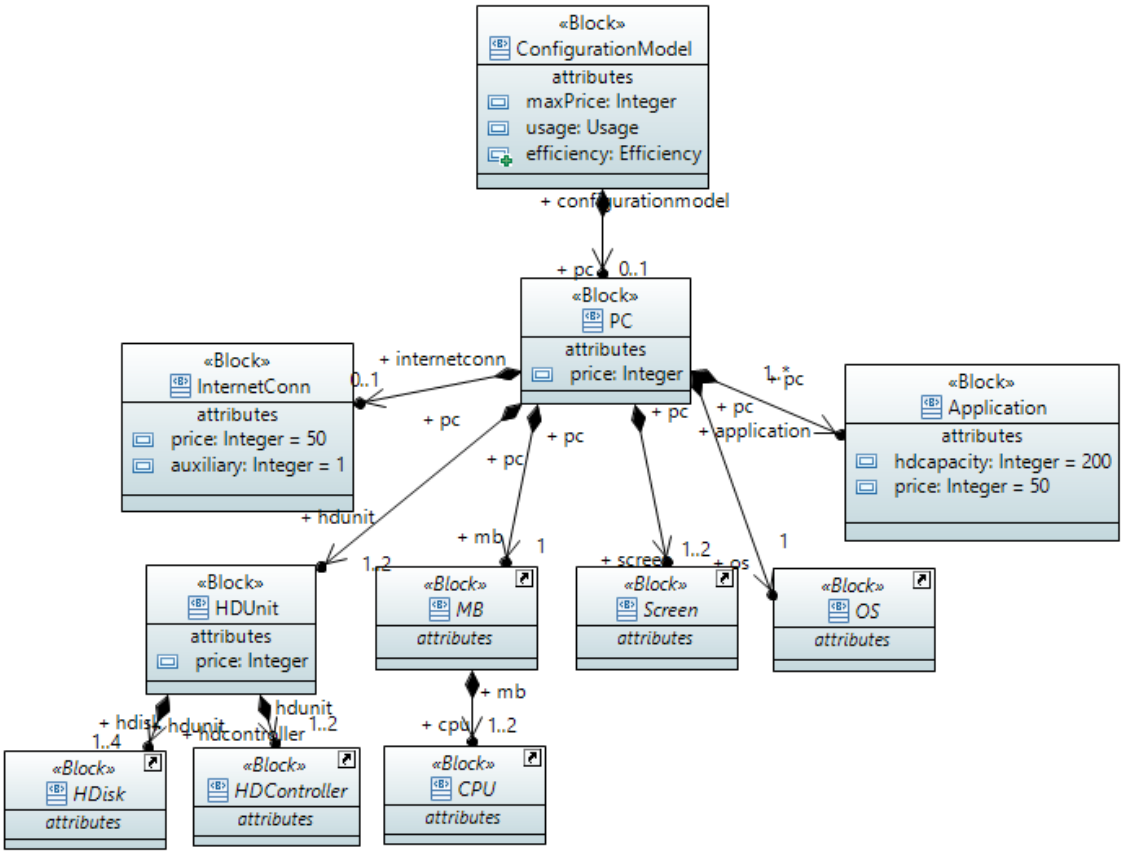


Figure 3.6.: Modelling Inconsistency Example: Block / Class Diagram in Papyrus (Source: Own Representation)

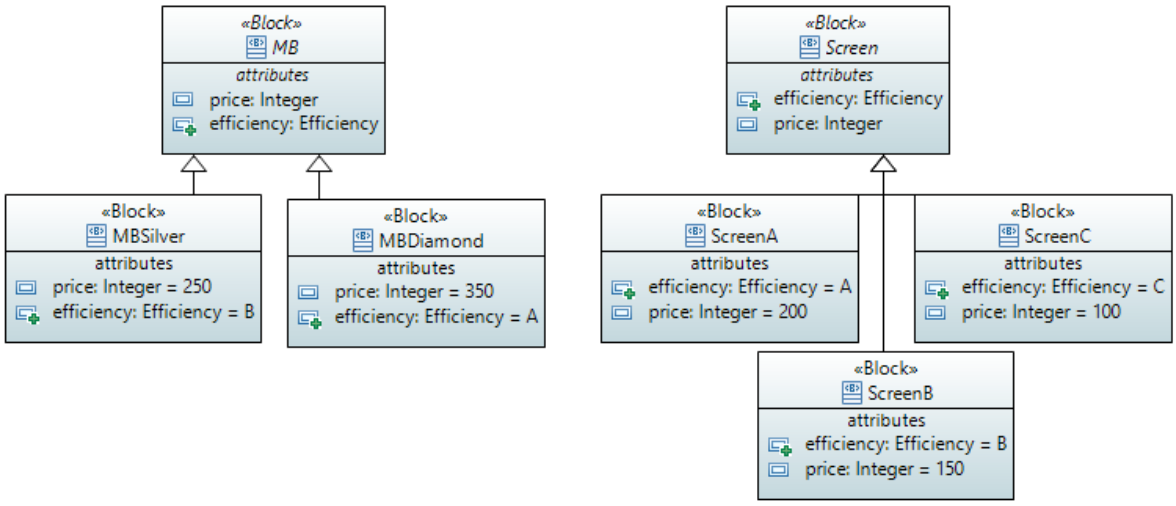


Figure 3.7.: Modelling Inconsistency Example: Block Diagram with inheritance arrow usage in Papyrus (Source: Own Representation)

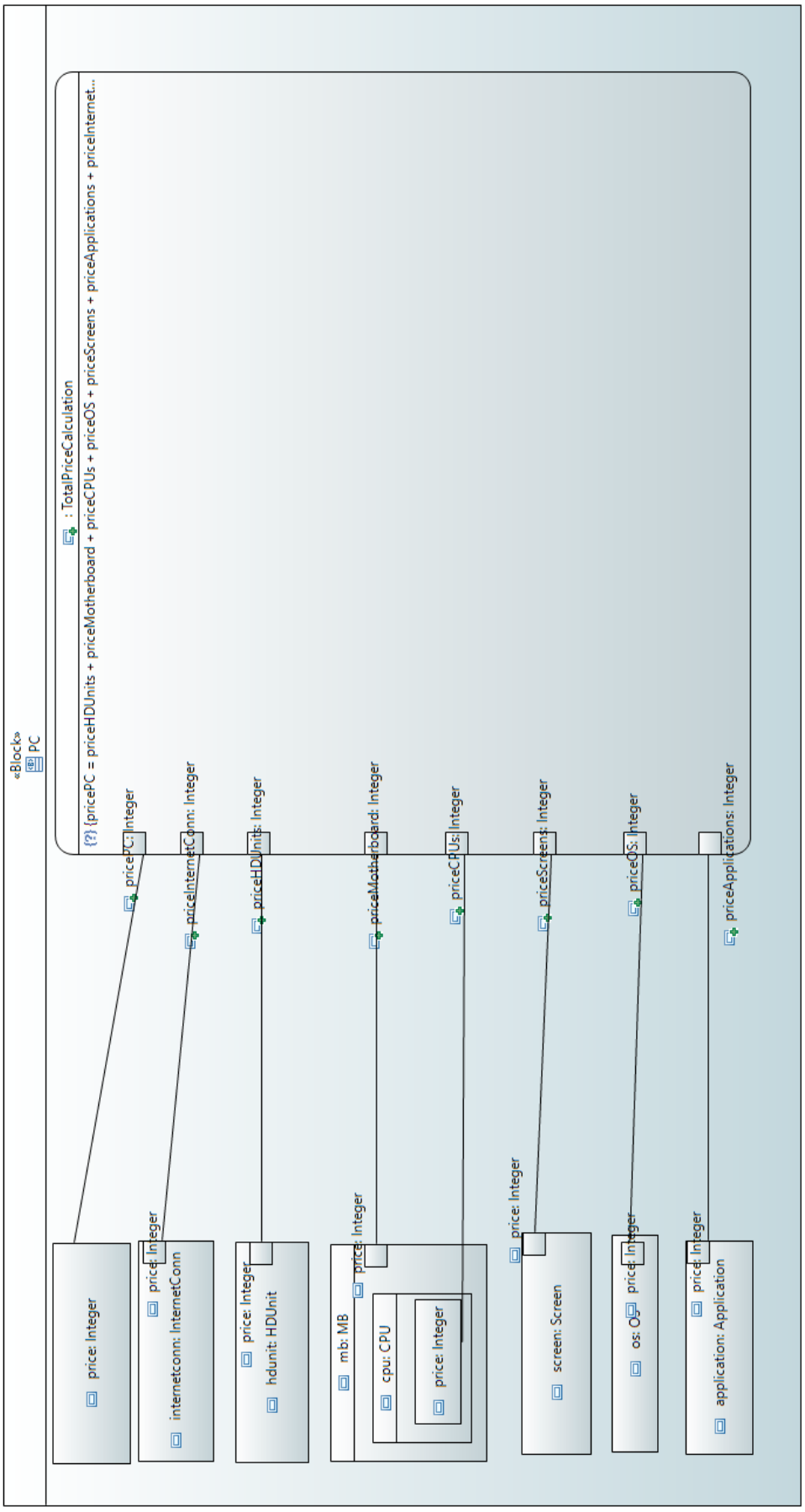


Figure 3.8.: Modelling Inconsistency Example: Parametric Diagram (Source: Own Representation)

The paper by Wong and Sun describes guidelines and criteria for the layout of UML diagram contents. Its contents are reiterated in a subsection of related literature 2.3.3 on page 24.
Applying the established guild lines and criteria:

- GC1 "Be selective": Was followed mostly for the properties and creation of sub-diagrams. However, as shown in figure 3.6, labels related to the arrows are always shown.
- GC2 "Use colours": No custom colour is applied. The elements have default colours applied.
- GC3 "Size consistently": The elements are not sized consistently. The elements with the same dimensions are default sized and thus not edited by the user.
- GC4 "Minimize crossings and bends": Were avoided.
- GC5 "Exploit proximity": Elements are pushed together. However, elements that are not logically connected are also placed close together.
- GC6 "Avoid overlapping": Overlapping can be observed in figures 3.6, 3.7, and 3.8 for labels and arrows. For parametric diagrams, in figure 3.8, Papyrus creates the label for the port (squares that are connected via a BindingConnector) on top of the box itself, which requires the user to move the label if it is too big before connecting the arrow.
- GC7 "Draw arcs orthogonally": All the arcs are drawn directly between elements.
- GC8 "Enhance flow": The order of flow was mostly followed. The 'start' element was put at the top of the diagram.
- GC9 "Orient labels horizontally": All labels are oriented horizontally.
- CC1 "Join inheritance arcs": In figure 3.7, the join was used partially, but the arrows overlap with the class shape.
- CC2 "Represent association": No association classes are used, only association arrows
- CC3 "Represent interfaces": No interfaces are present in the model. All generalizations are actual abstract types.
- CC4 "Place parents near children": Children are placed close to parents (see Fig.3.7).
- CC5 "Position superclasses above subclasses": Subclasses are placed below superclasses (see Fig.3.7).
- CC6 "Employ symmetry": Symmetry was tried to be implemented, but not on a pixel-accurate basis (see Fig.3.8).
- CC7 "Apply horizontal arcs for non-inheritance relationships": Figure 3.7 shows vertical or diagonal arcs used for directed-composition arrows.

These criteria are targeted for programs generating diagrams. Thus, enforcing some criteria is difficult or impossible in an editor-aided system based on simple rules.
Rules that are based on the content that is modelled are hard to aid for; these are:

- GC1: "Be selective" would require a more sophisticated approach to provide suitable filter options for elements based on domain knowledge. However, a warning could be issued when the number of properties for a class reaches a threshold.
- GC2 "Use colours" should be done by the user because unlike a generator, as it is used in Wong and Sun (2006), the editor does not have all the information to work with. An editor trying to colour elements could lead to accidental errors in understanding the model with limited knowledge. Weilkiens uses colour to differentiate the different layers of the model. Diagrams with elements from multiple layers remain their original colouring. (Weilkiens 2016)
- GC4 "Minimize crossings and bends" is hard to integrate into the editor automatically if the arrows should follow rectified paths as demanded by GC7 without moving the shapes the elements connects to. However, the functionality of "Arrange all" of Papyrus does a good job avoiding crossings.
- GC5 "Exploit proximity" has to be done by the user when graphical modelling, if an automatic layout is used the editor should help.
- GC8 "Enhance flow" similar to GC5 it is not known during modelling where the "start" is, if automatic layout is applied, the editor could help.
- CC4 "Place parents near children", same reasoning as GC5.
- CC5 "Position superclasses above subclasses", same reasoning as GC5.

Rules that the editor can aid with are:

a) GC3 "Size consistently"

b) GC6 "Avoid overlapping"

c) GC7 "Draw arcs orthogonally"

d) GC9 "Orient labels horizontally"

e) CC1 "Join inheritance arcs"

f) CC2 "Represent association"

g) CC3 "Represent interfaces"

h) CC6 "Employ symmetry"

i) CC7 "Apply horizontal arcs for non-inheritance relationships"

The selected criteria are explored in more detail and the current support of Papyrus is evaluated.

a) GC3 "Size consistently"

Papyrus partly supports consistent sizing of elements. Firstly, Papyrus creates elements with a default size, however, frequently the name of the element does not fit in the default dimensions causing the affected dimension to grow larger. An example of this behaviour can be seen in figure 3.6, the blocks "MB", "Screen", and "OS" are the default size but blocks "ConfigurationModel" or "InternetConn" require more space. Thus, quite often the sizing will be inconsistent. Secondly, an interaction for applying the same size of elements for one or both dimensions (width and height) exists by selecting both elements and selecting the option "Make same size" from the context menu, toolbar or pressing the hotkey.

b) GC6 "Avoid overlapping"

Can be aided by the editor by moving labels to avoid situations as seen in figures 3.6 and 3.8. The default arrows in Papyrus can create situations in which the arrow and its labels are on top of a class. This can be seen in figure 3.9.

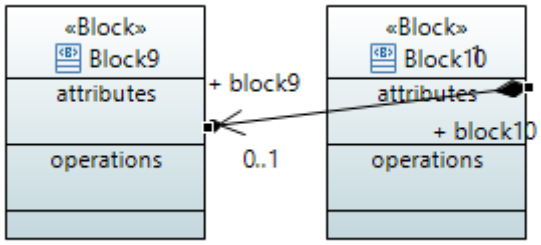


Figure 3.9.: Example of Default Arrows Overlapping after Creation in Papyrus (Source: Own Representation)

c) GC7 "Draw arcs orthogonally"

Draw arcs orthogonally is not the default behaviour in Papyrus, it rather connects the start and end of an arrow directly.

d) GC9 "Orient labels horizontally"

Labels are oriented horizontally by default in Papyrus.

e) CC1 "Join inheritance arcs"

This is not supported by Papyrus, the user has to move the start or end of the arrows to the same position and adjust the arrows. However, the editor could help with this criterion. When dragging an arrow to a new target while creating or editing, the inheritance arcs can be automatically merged. This is not done automatically by Papyrus.

f) CC2: "Represent association"

An association can be modelled in UML in two ways: a) the association should be displayed on the arrow edge itself, rather than b) a separate association class that is connected to the edge. The method of modelling recommended by CC2: "Represent association" is a). Papyrus already uses method a by default. Additionally, as Planas and Cabot found out in their user-test, Papyrus makes it hard to create an association conforming with method b) (Planas & Cabot 2020).

g) CC3 "Represent interfaces"

The goal of this criterion is to make interfaces look different from other class elements. The editor can help with this case by suggesting to the user that, if the interface in the current diagram has no elements, it can be replaced by the interface shape (a circle with no fill). Papyrus and other editors do not have such a feature.

h) CC6 "Employ symmetry"

This is not supported in Papyrus, besides alignment of two elements in their centre.

i) CC7 "Apply horizontal arcs for non-inheritance relationships"

This is a task for the user, but the editor could help by providing arrows that automatically connect horizontally if needed.
The topic of alignment is important to place elements on the same line vertically and horizontally. It aids the user to implement to GC3, GC4, GC5, GC8, CC4, CC5, and CC6. However, in the default version of Papyrus, neither the alignment guidelines nor the snap to shapes features are activated.

3.5. Quick Creation Pop-up

The quick creation pop-up (no official name exists for the element) is an element in Papyrus, see figure 3.10. It allows the user to create new elements inside a diagram. This is an accelerator for the creation process of elements. It allows faster creation compared to using the tree, property view or palette. The pop-up can be improved based upon the timings it appears. In practice, the pop-up is slow to appear, as it requires the mouse to not move for a while to show up. Additionally, the positioning of the pop-up compared to the mouse position requires the user to make precise movement to select a button inside the pop-up and avoiding moving too far away as this will close the pop-up again.

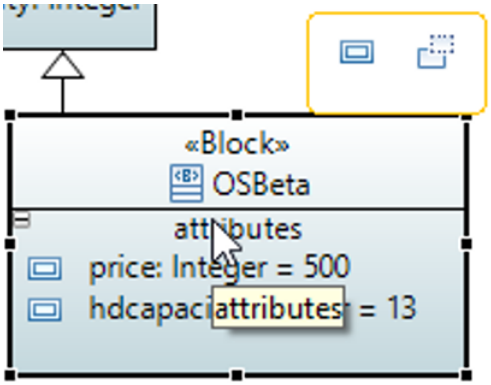


Figure 3.10.: Example of Quick Creation Pop-up in Papyrus (Source: Own Representation)

The resulting research question for modelling aid is: Can the editor aid in the diagram composition to improve usability by addressing the alignment of elements in the diagram, changes in arrow behaviour and adjustments to the quick creation pop-up?

3.6. Navigation

Navigation describes the collection of interactions that can be used to move through the given model and its diagrams. The primary navigation is done through the model explorer, which hierarchically displays most elements of a model in a file tree representation. Another navigation can be constructed by hyperlinking elements in diagrams to each other, for example a package in a diagram can link to the diagram describing its content. "Diagram 1" in figure 3.11 shows a package diagram with a nested package that can be clicked to navigate to "Diagram 2".
With the use of hyperlinks to navigate large diagrams, a habit evolved into a best practice that most diagrams have a linking to the diagram they were referenced in. The link to do this is applied to a comment in the top left of the diagram. For example, a diagram has at its root a package diagram showing its sub packages that link to the overview of the sub-packages, which in turn have the comment in the top left to navigate back to the root overview. Figure 3.11 features such a comment in "Diagram 2" to navigate up towards the root of the model, when it is clicked it navigates the user to "Diagram 1".

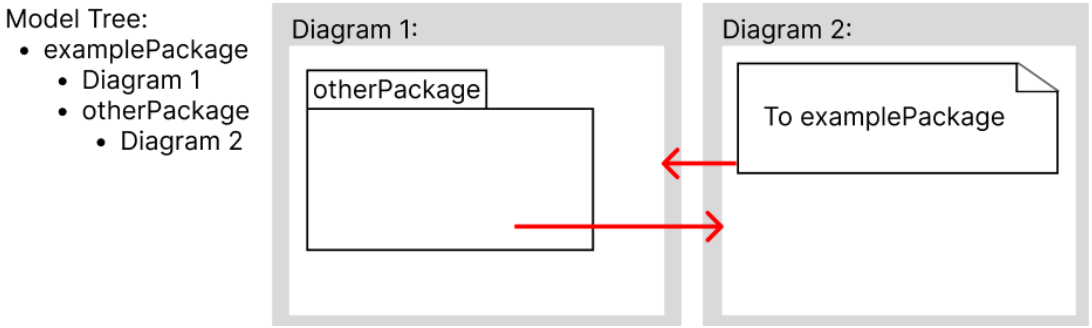


Figure 3.11.: Simplified Example of "Navigate Up" Comment Navigation (Source: Own Representation)

Both of the introduced types of links share the problem that Papyrus does not provide an

indication that an element in the diagram has a link on it, which can be navigated. The user has to interact on their best assumption that they can navigate by double-clicking the specific element. If no link is present, a pop-up window opens to create a hyperlink, figure 3.12 shows the popup. In theory, opening a form to create the missing thing follows the guidelines that Nielsen and Molich defined in his 10 usability heuristics (see 2.2.1) to provide the user to recover from an error. However, if this window is opened accidentally, it is counterproductive in terms of UX as the user does not want to create a link and might get confused what to do in the pop-up. The frequency with this pop-up appears is pretty high because other interactions that require double-click (edit of element's name/text) and single-click (move (drag) or select element) exist on the same element.

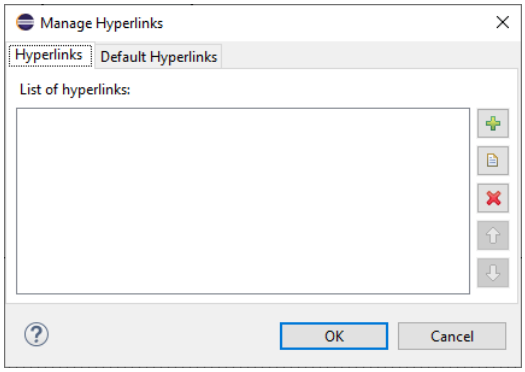


Figure 3.12.: Papyrus Hyperlink Pop-up (Source: Own Representation)

Furthermore, the best practice link to get back to another diagram has two additional problems. Firstly, that it has to be created manually and secondly, if a diagram is bigger than the viewport (see V) the link-up can move outside the viewport and is no longer visible.
The resulting research question for the third part of navigation is: How can the visibility of navigable links between diagrams and the best-practice upwards link be improved, resulting in less friction for a user and limited setup on the side of the user?

4. Concept

This chapter introduces and applies the concepts to achieve the goal of improving usability. The first section (4.1) discusses general modifications spanning the different primary goals. The following three sections (4.2, 4.3 and 4.4) address this thesis's main goals, and the remaining sections explain the concepts applied to components that are part of the editor.
For this prototype, the design principles for good UX should be adhered to, and thus the concept tries to follow the established theoretical frameworks in chapter, 2. The developed prototype is named "Modelling Studio".

4.1. General

4.1.1. Material Design System

The first general change is the introduction of a Material Design system to the prototype. The advantages of using a material design system are that:

- the plethora of existing components that can be reused
- the components follow UX guild-lines
- components are adaptable with providing a theme
- custom components can be created following the underlying design principles
- a widely used system feels familiar to a user (Raskin 1994)

In this thesis, the Google Material Design System (see Google (2022a)) is used. This system is used widely on desktop, web, and mobile phones (mostly Android).
A fundamental principle for spacing and sizing in the system is the "8pt-Grid Rule" (see 2.2.3). The spacing and sizing amounts are extracted from this rule in the following sections.

4.1.2. Introduction of modes

A bigger diagram area can help to expose more of the diagram content without hiding elements outside the viewport. While the user views the model with its diagrams, parts of the editor like the palette or actions to modify diagrams are not utilized. In some instances, it is also not preferred. While viewing, for instance, a user does not want to change the diagram accidentally. Thus, a switch between viewing and editing mode was introduced, limiting possible interactions. For example, hiding the palette and preventing an element in the diagram to be created, moved, resized or deleted.

4.1.3. Not Implemented Interactions

As this thesis produces a prototype, not all features will be implemented or present. It is important, however, that if such an unimplemented interaction happens, the user receives feedback. If not, the user could get frustrated by the lack of visibility of the system status (Nielsen 1994).

4.2. Hierarchy

As mentioned in the problem definition, 3.2, the positioning of the parts of the editor can be optimized. Figure 4.1 shows a simplified version of the placement of the editor parts for Papyrus.

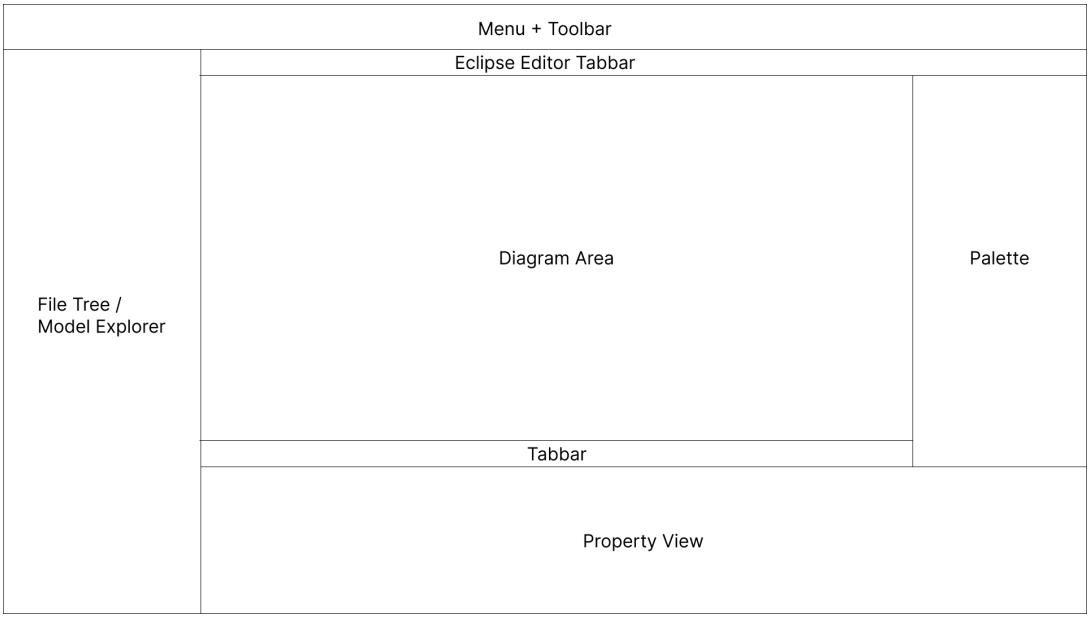


Figure 4.1.: Papyrus Simplified Representation of Editor Parts (Source: Own Representation)

The positions of the 'property view' and the palette are switched to improve the overall layout. The resulting hierarchy for modelling studio is depicted in figure 4.2.

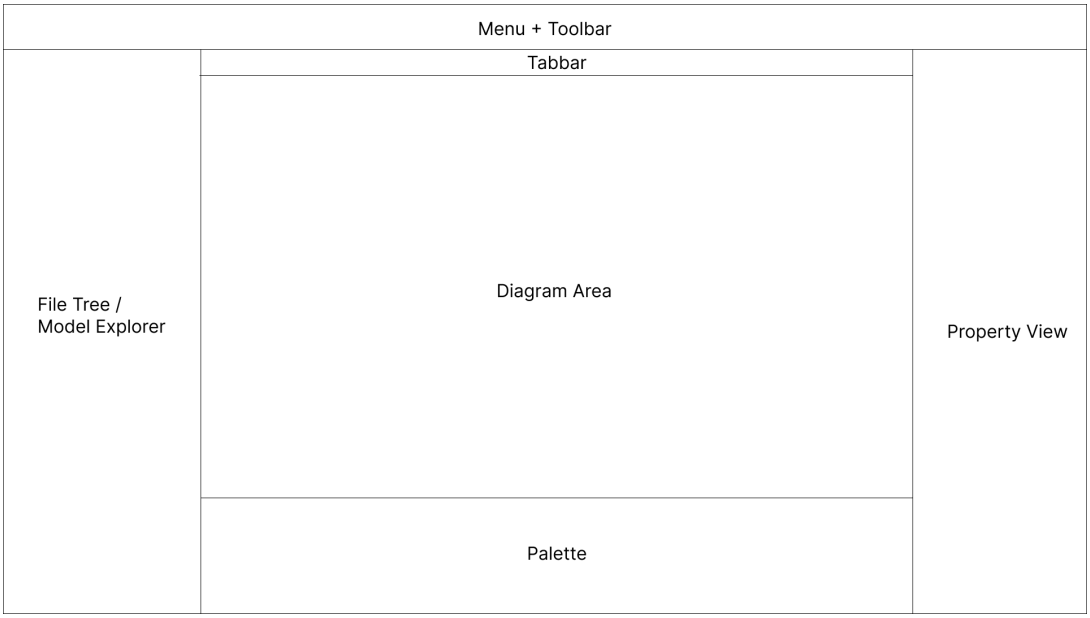


Figure 4.2.: Modelling Studio Simplified Representation of Editor Parts (Source: Own Representation)

By moving the properties view to the right of the application, the inputs of the properties can be stacked on top of each other, thus using the vertical space available while requiring less horizontal space. Furthermore, placing the labels above their connected input field minimizes additional horizontal space. Matteo Penzo performed a usability study, finding that bold labels above the inputs are the best for fast completion time of tasks (Matteo Penzo 2006).
The vertical stacking of inputs also allows the user to use the 'infinite edge' on the right side of the monitor to click into the input for an element's property.
Not only the property view is expected to benefit from this change, but also the palette. The palette before was a friction point to use for users, with the relocated palette more horizontal space is available in which columns can be created for the different categories of items that can be inserted into a diagram. By sorting the categories internally by the most used item overall (no changes in the product) the palette does not have to be extended all the way to be used efficiently.
With the potential in size decrease in the palette's and 'properties view's' contributing dimensions, the diagram area naturally grows larger, allowing a bigger diagram to be shown.
Figure 4.1 shows a component called "Eclipse Editor Tabbar". In this tab-bar the current editors of the IDE are shown. As mentioned in 3.2 the duplication of a tab-bar is not necessary, and therefore the modelling studio version only has one tab-bar at the top of the application. The positioning of the tab-bar at the top is recommended by Jakob Nielsen's "Tab-Usability Guidelines" (Jakob Nielsen 2016) and Google's Material Design System (Google 2022b: section "Placement").
The hypothesis of the hierarchy changes is: H_0 : Users are rating the changes in hierarchy in the parts of tab bar, property view, and palette higher in Modelling Studio than Papyrus, while the other parts show no decrease in rating of the element. H_1 : Users provided decreased rating from any of the changed parts in hierarchy or the other parts in Modelling Studio compared to

Papyrus.

4.3. Aiding Diagram Composition

A general change is made for the concept that addresses 3.4 a) GC3 "Size consistently". Papyrus offers good tools to fix different sized elements, however the issue of small default sizes can be improved by increasing the default width of the elements, so it is less common to need to grow larger.
For aiding the diagram composition, three separate categories are addressed: 4.3.1 Diagram Alignment Helper, 4.3.2 changes to default arrows, and 4.3.3 the changes to the quick creation pop-up.
However, some criteria the editor could help with are not dealt with in this thesis.
The proposition of 3.4 g) CC3 "Represent interfaces" for the editor to suggest changing the appearance of an interface, while it is useful to adhere to the recommendation, neither this concept nor prototype implement such a feature.
Moreover, 3.4 f) CC2: "Represent association" is not incorporated into the concept, as the prototype does not support association classes.

4.3.1. Diagram Alignment Helper

The diagram alignment helper is a system of guidelines providing points to align elements with each other. When an element is being moved, guidelines should be drawn inside the diagram area displaying alignment possibilities.
An element inside the model should have 3 default guidelines with which the moving element can be aligned against in vertical and horizontal direction. Figure 4.3 shows the guidelines of a static element.

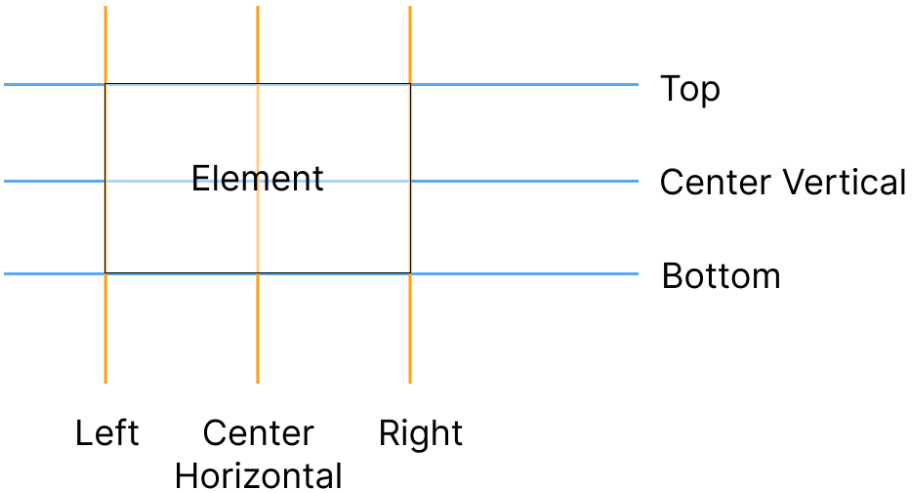


Figure 4.3.: Names and Positioning of Guidelines per Element (Source: Own Representation)

The guidelines have a snap functionality, this means if the mouse cursor moves close to a shown guideline the element being moved snaps to the guideline from multiple pixels away. To provide

the user with more information during alignment of elements, the alignment line switches colour to green if the element is aligned properly.
However, displaying all the lines in the diagram with multiple elements can be overwhelming for the user. Thus, the number of lines is filtered based on the movement direction of the element. For example, figure 4.4 shows the movement of an element from the right to the left. In part 1 the moving box is to the right of the element and only the left most guideline of the static block is displayed. The reason, why only the left guideline is displayed is based on the law of continuation because the other guidelines would not produce a continuation directly. The centre guideline is displayed shortly before the centres of both boxes cross, see part 2. Although, it goes against the continuation, the centre allows CC6 "Employ symmetry" to be supported for the editor. Additionally, a common use-case is that the user wants to position subclasses below their superclasses (CC5) and centre them (CC6 "Employ symmetry"). To support this interaction, all subclasses could be selected, joining their bounding shapes to a single box and moving it in the centre below the superclass. Part 3 and 4 show that an aligned element is indicated by green guidelines.

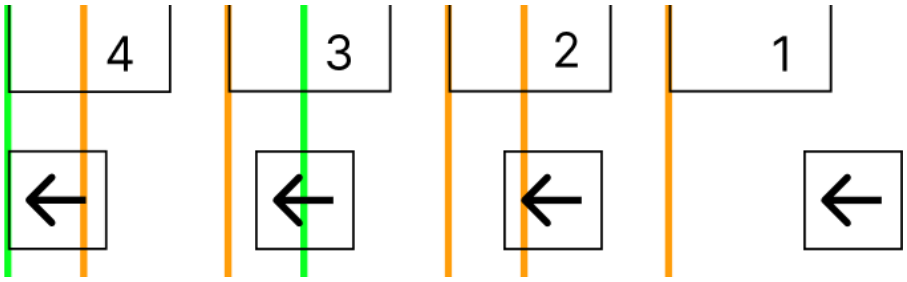


Figure 4.4.: Guideline Behaviour During Movement (Source: Own Representation)

Sizing of elements can also be aided by guidelines. If an element is being resized, the guidelines in the direction of expansion are displayed.
A big part for realizing CC6 "Employ symmetry" is the spacing between elements. Additional information on the space between elements is needed while moving to create same sized spaces between elements. The spacing in pixels between two aligned elements should be displayed when the element is dragged. As this is a feature that might not be used by most users, the numbers should not be distracting and be less noticeable. Figure 4.5 shows an example of the spacing indicator. The orientation of the number is always horizontal, following GC9 "Orient labels horizontally".

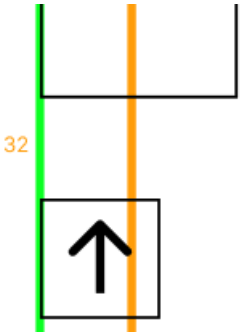


Figure 4.5.: Spacing Number on a Guideline (Source: Own Representation)

4.3.2. Default Arrows

Further refinements to the diagram composition are possible through changes in the default arrows.
When creating a new arrow, instead of connecting the start and end position like Papyrus.
A new arrow is created between the best anchor point side for the source and target. This way, the arrow is allowed to choose the side it connects to itself, avoiding overlapping (GC6) in the progress. A layout, for example, that connects an arrow from the south of an element with the east of another requires the arrow arcs to be not orthogonal any more. Therefore, all default arrows in the concept are following an orthogonal arc if needed (GC7). If an arrow connects elements that require two adjustments in the arc of the arrow, the orthogonal arcs will adhere to CC6 "Employ symmetry" by splitting the dimension opposite the anchor direction in half. If the anchors are connected between the north and south and the elements are horizontally offset, three arcs are needed to make an orthogonal connection. In this case, the vertical space between the anchors is split equally to ensure symmetry.
Addressing changes to specialized arrow types: Firstly, for generalizations, the arrows are centred on the superclass by default, resulting in a merged generalization arrow head if multiple exist, fulfilling CC1 "Join inheritance arcs". Furthermore, association arrows have reduced label information, as in practice, actor and role information of the relationship is not needed (GC1 "Be selective"). With less text for labels, overlapping is less likely, thus improving GC6 "Avoid overlapping".

4.3.3. Quick Creation Pop-up

Changes to the Quick Creation Pop-up focus on the way the pop-up is opened. Instead of a delay-based appearance, the system should become more deterministic by moving the functionality into the context menu. Moreover, the pop-up appears instantly after right-clicking. The Papyrus pop-up does not support quick creation of arrows. This is added in the concept by adding a context menu entry named "Arrow to". It starts the arrow at the current block and awaits a second click to finish the arrow.
The hypothesis is as follows: H_0: Providing help with alignment and changes to default arrow based on best practices for UML diagrams allows the user to create diagrams closer to the best practices.

4.4. Navigation

Navigating through a diagram should contain the same features that Papyrus offers and improve upon them. These features are:

- Open diagrams through the explorer
- If no diagram is open, offer a "welcome" page with a list of existing diagrams
- Allow links on elements
- Interacting with a link navigates to a different diagram in the model

The "welcome" page of Papyrus offers good usability and has not to be adjusted. Thus, Modelling Studio features a "welcome" page with similar information, an example of it can be seen in figure 4.6. The only difference is that the prototype does not support Internationalization (i18n) language selection which is presented on the left side of the page in Papyrus, however in the used example models no i18n configs besides default are used, i.e., the list is empty.

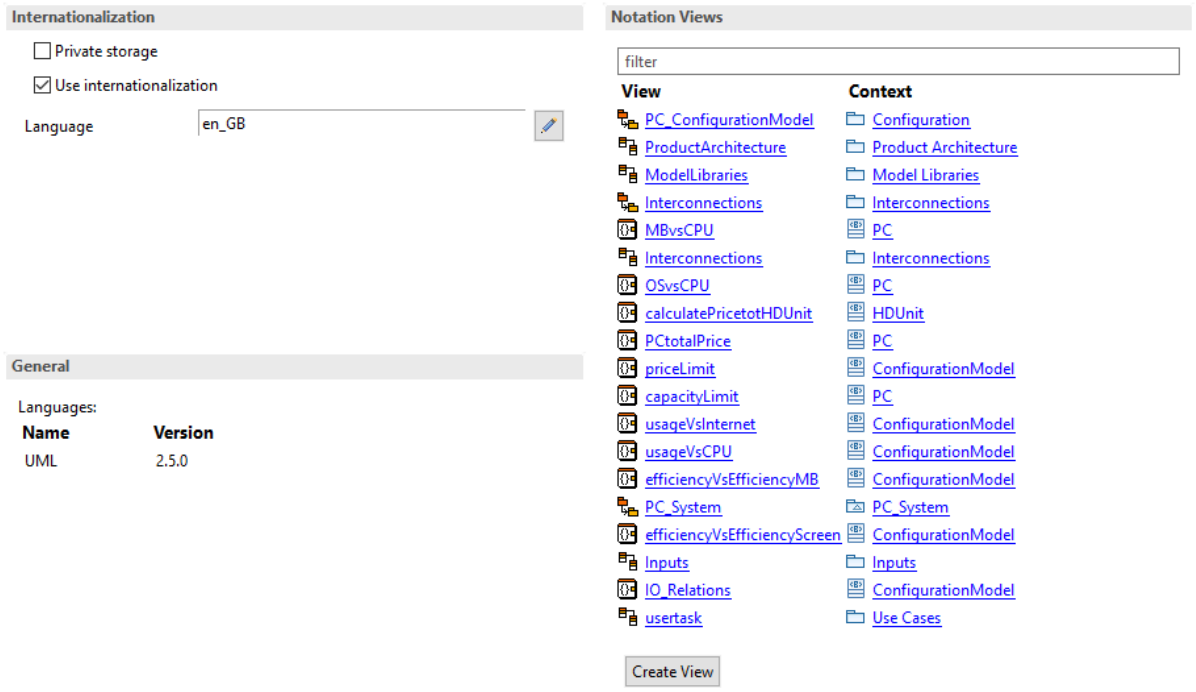


Figure 4.6.: Example for 'Welcome' Page (Source: Own Representation)

Furthermore, the possibility of links is added to the concept, but the way these are opened, displayed and created is changed. Firstly, elements with a valid link gain an icon indicating that a link exists and can be followed/opened. Secondly, navigating via a link happens on a single click on the link indicator or a press on the class shape with "CTRL" pressed. As mentioned in 3.6 the double click action is overloaded. It is used for following links, adding links if they are missing, edit of elements fields, and selection of sub-elements (e.g., a property of a class). After the change, the double click will always open the element in the 'properties view', even if the 'properties view' is hidden at that moment. Creation of a link is a less common action, and thus it was moved to the context menu for the element. This is also the same position for the alternative interaction to create, edit, and delete a link in Papyrus.
In 3.6, the best practice in SysML of having a link to the diagram in the higher layer is introduced. This best practice can be converted into a feature of the editor by reserving a button for the link to the other diagram instead of a custom comment. By creating a button in the editor, the link is always visible, unlike a comment with a link in a diagram that can be scrolled to hide itself. For a user, this means less mouse scrolling, allowing muscle memory to build for a fixed location in the editor, and be given a reminder to configure the link (aka. follow the best practice).
The new button is positioned inside the diagram area on the top-left. It is below the tab-bar, one of the primary ways to navigate the diagram.

Lastly, a commonly used component for improving navigation are breadcrumbs. (Laubheimer 2018; Nielsen 2007) Nielsen summarizes their findings about breadcrumbs as: "One line of text shows a page's location in the site hierarchy. User testing shows many benefits and no downsides to breadcrumbs for secondary navigation" (Nielsen 2007). Figure 4.7 shows breadcrumbs for an example model. The circled number 1 shows the current diagram that is opened, and 2 the package the diagram is in, and continues for all the ancestors in the hierarchy until the root, named "workspace", is reached. Navigation with breadcrumbs works in two ways. Firstly, clicking a text (e.g., indicated by 1 or 2) shows a drop-down selection for the sibling objects of the currently selected object. Secondly, clicking a chevron (e.g., indicated by 3) opens the drop-down selection with the children of the element to the left of the chevron. In the case of the prototype only diagrams are able to be navigated to, thus unless a diagram is clicked the next child menu should be opened to reduce clicks needed. Nielsen claims that "[b]readcrumbs never cause problems in user testing: people might overlook this small design element, but they never misinterpret breadcrumb trails or have trouble operating them" (Nielsen 2007). As the breadcrumbs are the purpose of a secondary navigation and do not represent an integral part of the editor, their appearance should gain minimal visible-importance. The user should not be distracted by the element.

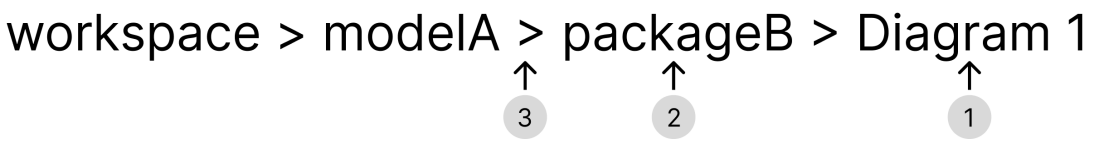


Figure 4.7.: Example of Breadcrumbs (Source: Own Representation)

The hypothesis for the proposed changes is that: The general usability of navigating through a model can be improved:

- By applying an icon to a valid link on an element, the link becomes more visible compared to Papyrus
- By removing the double click interaction on elements for link interaction the user is less frustrated
- By integrating the best-practice button into the editor the visibility for the link increases
- By providing breadcrumbs a secondary navigation is added that is not distracting and is predictable in navigation

5. Implementation

The chapter at first introduces the used technologies and programming languages, 5.1, secondly implementation specifics are explained, 5.2. Figure 5.1 shows an example of a diagram inside the created editor Modelling Studio.

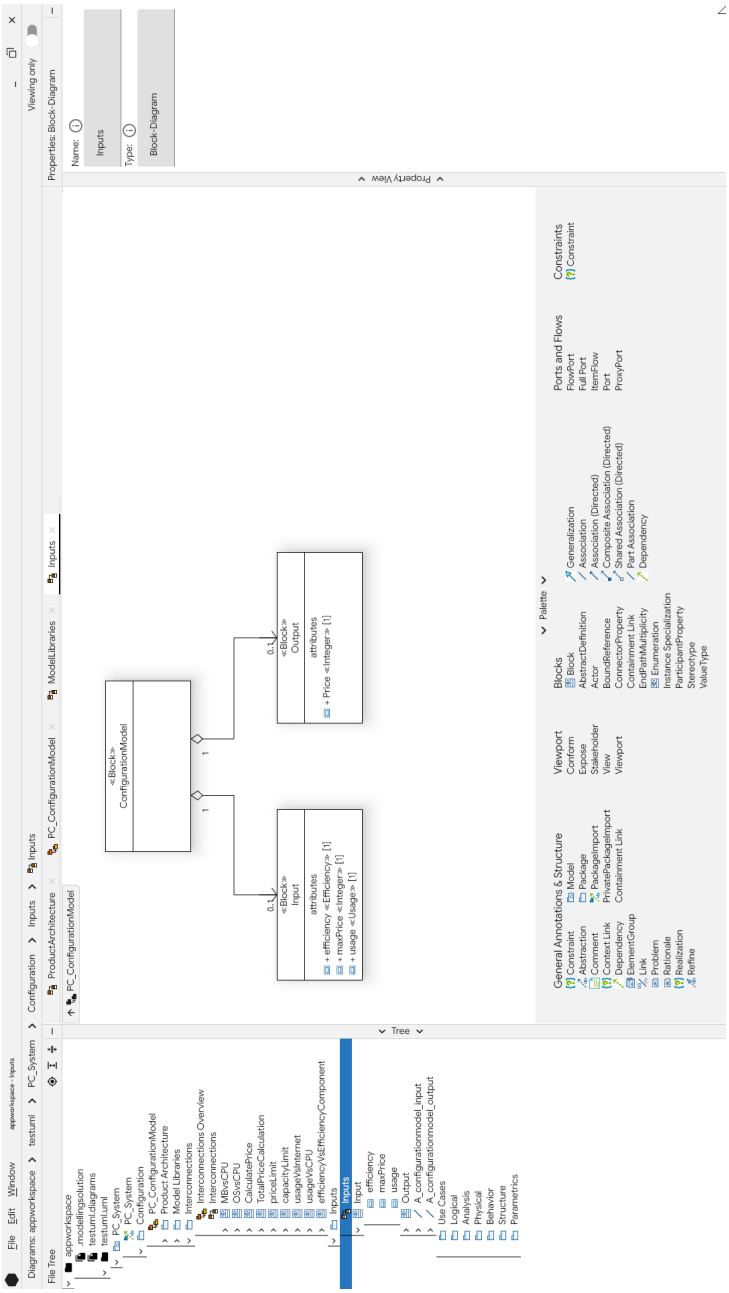


Figure 5.1.: Example of Diagram in Modelling Studio (Source: Own Representation)

5.1. Used Technologies

The section describes the technologies (modules / frameworks) and programming languages used during the implementation of the application.

Kotlin

Kotlin developed by JetBrains and Open-source Contributors is the primary programming language of the prototype, as it is used to implement most of the business logic as well as the complete front-end / UI code. The latest version, 1.6.20, available at the start of the thesis, was used. (JetBrains & Open-source Contributors 2022) "Kotlin is an open-source, statically-typed programming language that supports both object-oriented and functional programming. Kotlin provides similar syntax and concepts from other languages, including C#, Java, and Scala, among many others" (Android Developers 2019).

Kotlin Coroutines

Kotlin Coroutines is the language's module for asynchronous programming. In this project, it is used to move computation into a different thread to make sure the main thread that handles the UI updates does not freeze and skips frames. Coroutines are also used by the UI framework to create animations or emit side effects.

Java

Java, developed by Gosling and Sun Microsystems, is the second programming language used in the prototype. (Gosling & Sun Microsystems 2021) The framework EMF (see below) has open-source examples in Java, due to the interoperability of Kotlin with Java the code remained in Java.

Jetpack Compose

"Jetpack Compose is a modern toolkit for building native Android UI. Jetpack Compose simplifies and accelerates UI development on Android with less code, powerful tools, and intuitive Kotlin APIs" (Android Developers 2022). The toolkit was chosen due to prior experience, and it represents a state-of-the-art UI toolkit. Additionally, a near-complete implementation for the chosen Material Design framework (4.1.1) is included.

Compose Multiplatform

Compose Multiplatform is developed by JetBrains. It enables the usage of Jetpack Compose as it implements a full support on multiplatform. For the prototype, relevant targets of multiplatform are the desktop targets (e.g., Windows, Linux, and Mac). (JetBrains 2022)

Smaller Compose Libraries

Other libraries that are connected to the UI framework Compose are:

- Compose Desktop Template by theapache64 (2021) as a basic setup for View Models, Dependency Injection with Dagger (Google & Square 2022), Navigation, etc.
- Decompose developed by Ivanov (2022) for reliable navigation between screens.
- JetBrains Jetpack Compose Slidepane provided by JetBrains (2022: as seperate library at components/SplitPane/library) for on-mouse-drag resizeable areas inside the editor.
- Arrow Core created by Arrow (2021). It brings functional programming functionality for Kotlin, it is used for validation purposes in the prototype.

UML Libraries

The following libraries are used to load, edit UML from files.

- Eclipse EMF / ECORE by Eclipse Foundation (2021). It provides the basis for interactions on resource sets which contain the formalized UML model and can be interacted with. The interactions can be managed through transaction with Eclipse Modelling Framework (EMF).
- EMF Cloud - Modelserver by Eclipse Foundation (2022) allows EMF interactions to models through a Representational State Transfer (REST) Application Programmable Interface (API)

Gradle KTS

Gradle is a build tool for multiple languages, for example, Java, Kotlin or Groovy. It is developed by Hans Dockter et al. (2022). In this prototype, both the Groovy and Kotlin (KTS) primer were used.

5.2. Implementation Specifics

The implementation can be viewed on GitHub, redirected from `http://msc-code.mayr.fyi`.

5.2.1. Structure

The repository is split into different Grade modules, their packages and description:

- “canvas” Canvas Implementation
- “combined” Main UI of the application that also combines all other modules to the product
  - “di” Dependency Injection Component, Module
  - “model”
    - “command” Commands that are used in the application
    - “mock” Content that was mocked
    - “notifications” A notification system

        * “validation” A validation system for inputs
        * “TMM” Tree Meta Model (TMM) provides an observable representation for the underlying ECORE model
    - “ui”
        * “adjusted” Adjusted implementation for a diagram area based on module "canvas"
        * “components” Components used in the UI
        * “feature” Different features of application: debug windows, main, splash screen, wizard
        * “navigation” Router implementation
        * “uml” Classes that handle diagram content like UML diagram loading from file, structure definition and rendering
        * “value” Values for Theming, Spacing, Typography, Colours, and Tooltip Texts
    - “util” Utility functions
    - “App” Entry point for the whole application
- “ecore” Interactions with EMF and Modelling Server
- “forked-libs” contains libraries or files from them that were used, but source code was adjusted
- “recorder” code that records mouse movements and key presses for the application

5.2.2. Undo / Redo System with Commands, Jobs and Progress Indications

Starting off, the interaction of undoing a prior action requires an undo / redo system. Such a system may be built using the memento pattern or the command pattern. For this prototype, a command pattern was chosen because it offers more freedom in the interaction implementation (potential of queueing, known to be compatible with parts of the frontend framework) and requires less memory and computational resources from the application. (Erich Gamma, Richard Helm, Ralph Johnson, & John Vlissides 1995) The command is defined by the interface “ICommand” that can be seen in code fragment 5.1.

```
interface ICommand {
    fun isActive() : Boolean
    suspend fun execute(jobHandler: JobHandler)
    fun canUndo() : Boolean
    suspend fun undo(jobHandler: JobHandler)
    fun pushToStack() : Boolean = true
}
```

Code Fragment 5.1: Command Pattern: Command Interface

Following the pattern, the command has query methods for checking if the command can be activated and if it was activated once one if it can be undone. If the corresponding query method returned true, the "execute" or "undo" method is invoked. An addition to the normal pattern is the last function "pushToStack", it determines if the command is put on the stack after the first execution.
The command receives a "JobHandler" object as parameter for the "execute" and "undo" functions. It is used to push a job to the executing part of the application, if an execution of a job takes more than 200 millisecond the UI will show a progress indication to let the user know of the current state. The progress indication supports tick-based (for measurable progress) and infinite progress (unknown duration progress) states, if known a message can be added.
However, in practice the progress bar is rarely seen in action (actually never in the usability test) because most actions are over before the 200 millisecond threshold.
Figure 5.2 shows the progress indication in the editor, and figure 5.3 shows the pop-up that can be opened by clicking any of the progress indications in the editor.

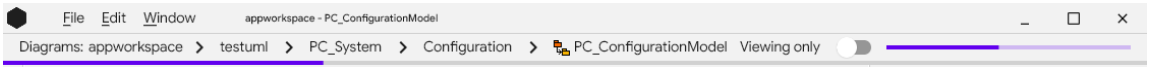


Figure 5.2.: Example of Background Progress Indication in Modelling Studio (Source: Own Representation)

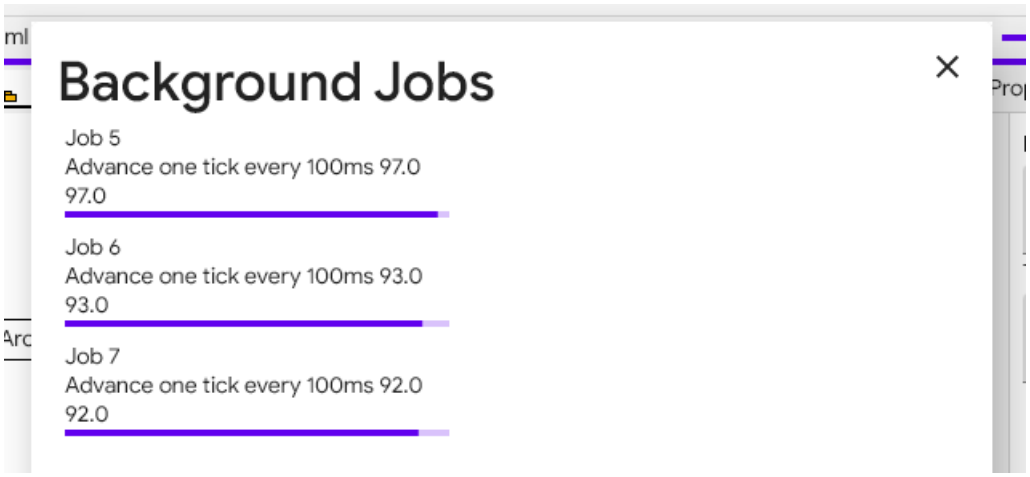


Figure 5.3.: Example of Background Progress Indication Pop-Up in Modelling Studio (Source: Own Representation)

5.2.3. Data Model

From the loaded ECORE model (aka. the UML/SysML model) and the separate data for created diagrams, a Tree Meta Model (TMM) is created. For most elements from UML/SysML a typed TMM version exists. The TMM object acts as a Proxy for the actual object from the model. The wrapping of the UML/SysML elements is done to, firstly, enable the usage of Kotlin's type system instead of Java's, secondly, assure that all changes to UML elements are properly observed and propagated to listeners, and lastly, allowing other UI state to be in cooperated.

5.2.4. Components

Menus

The menus support mnemonic shortcut. A mnemonic shortcut is activated by pressing Alt and the underlined part in the name of the menu that should be opened. Figure 5.4 shows the left side of the menu bar.

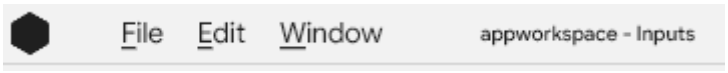


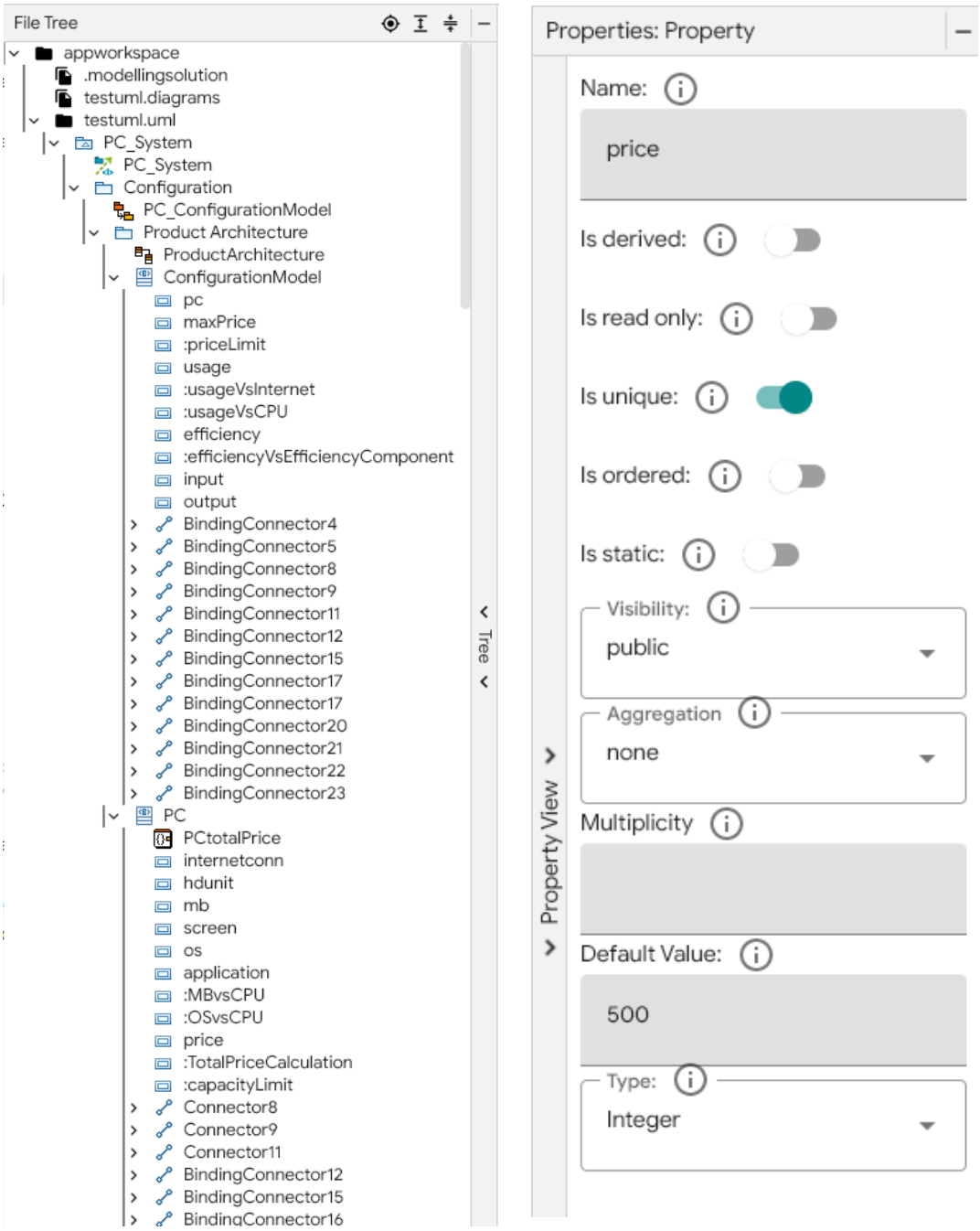
Figure 5.4.: Example of Menubar in Modelling Studio (Source: Own Representation)

File Tree / Model Explorer

The file tree displays the structure of a given TMM. However, the TMM represents the whole model and the explorer should only show elements that are currently expanded. To achieve this, the component creates a separate tree with only the visible elements. Interactions on the tree are forwarded to the TMM element if not handled by the component itself. Figure 5.5a shows an example of the component.

Properties View

The property view was implemented following the concept, inputs and labels are stacked on top of each other. Figure, 5.5b shows the property view for a selected element.



(a) File Tree / Model Explorer

(b) Property View

Figure 5.5.: Examples of File Tree / Model Explorer and Palette in Modelling Studio (Source: Own Representation)

Palette

The palette uses the complete horizontal space to display the elements. The elements are organized in the same categories as in Papyrus, but sorted by the importance to the modelling process. Figure 5.6 shows the palette for a block-definition diagram.

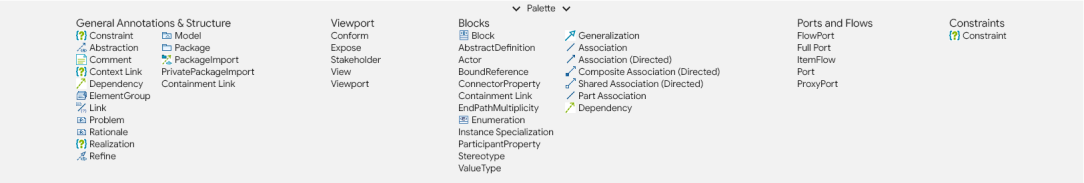
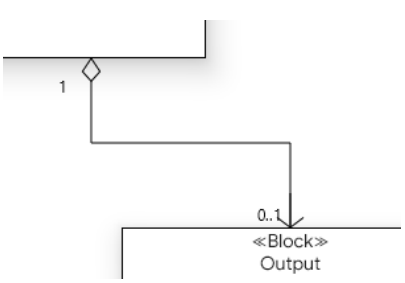


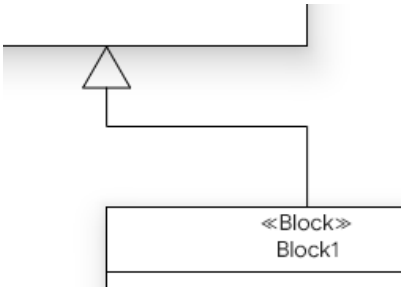
Figure 5.6.: Example of Palette for Block-Definition Diagram in Modelling Studio (Source: Own Representation)

Arrows

The arrows select their path between the two selected elements, they determine which cardinal side to connect the elements from and how the arcs are positioned. If an arrow path cannot be a vertical or horizontal line, a path is split in 3 parts, which are orthogonally at their connection points. Furthermore, the arrows try to create the middle element half-way between the elements to create a symmetry in the arrow path. Figure 5.7 shows examples of a composite arrow in 5.7a and in a generalization arrow in 5.7b.



(a) Composite Arrow (Shared)



(b) Generalization Arrow

Figure 5.7.: Examples of Composite and Generalization Arrows in Modelling Studio (Source: Own Representation)

Guideline

The guidelines are implemented as shown in the concept, see figure 5.8 for an example alignment of a block.

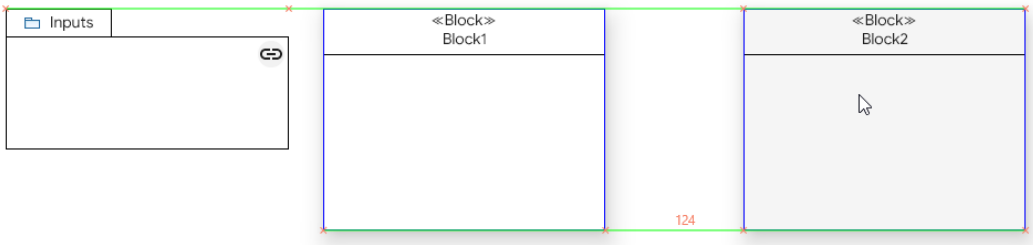


Figure 5.8.: Example of Guideline in Modelling Studio (Source: Own Representation)

Link

The link is indicated by a chain icon on the top right of the element, see figure 5.9a. If the mouse pointer is hovered above the link, a text with the name of the destination is displayed,

see 5.9b. The text can also be expanded if “CTRL” is pressed.

(a) Link in Short Version (Only Icon)

(b) Link with Expanded Text

Figure 5.9.: Examples of Link Visualization in Modelling Studio (Source: Own Representation)

Navigate Up Diagram

Figure 5.10 shows an example for the navigate up button. The button has an icon for the type of diagram for the link’s target.

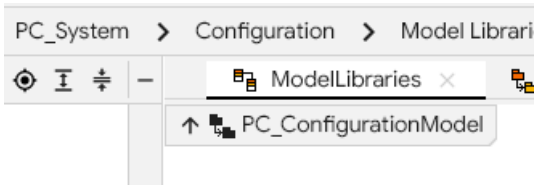


Figure 5.10.: Example of Navigate Up button in Modelling Studio (Source: Own Representation)

Quick Creation Pop-up

The quick creation pop-up is moved into the context menu, figure 5.11 shows an example for the context menu on a property.

Figure 5.11.: Example of Quick Creation Pop-up in Modelling Studio (Source: Own Representation)

6. Evaluation

This chapter introduces the methods used to test, the results from the user test.

6.1. Methods

The evaluation will consist of two main parts: firstly, a user-test with the developed prototype Modelling Studio and Papyrus, and secondly, a questionnaire.

6.1.1. User Test

For the user test, a within-subjects design is used. A within-subjects or repeated-measures design for a study tests a single individual on all the conditions of the test (Raluca Budiu 2018). This means that each individual is tested in both Modelling Studio and Papyrus. One of the limitations of with-subject designs is that respondents may feel pressured to provide answers that match their own or the experimenter's expectations (Charness, Gneezy, & Kuhn 2012). Rosenthal, R. states that both hypotheses and expectations of an individual of another person or thing have an effect on the accuracy of interpersonal predictions (Rosenthal, R. 1976). This implicates the research method of within-subjects design by individuals transferring hypotheses and expectations to the second application that should be tested. Rosenthal, R. continues, "[i]f we simply ascertain people's expectations of others' [behaviour] and correlate these with the others' subsequent [behaviour], the two components of experiential accuracy and self-fulfilling accuracy will be confounded" (Rosenthal, R. 1976: p.408). To balance this effect between the editors, two test groups are created. The distinguishing difference between the groups is the order in which the editors are tested and the tasks that they are asked to complete in a specific editor.
Table 6.1 show the groups and the order of editors. By switching the editor among the groups, the tasks (in table: 'Questions' and 'Model') that are asked are also tested for each editor. Both editors get different tasks for each scenario to minimize the amount a person can remember answers or thinks they can. The difficulty of the tasks is also asked to assure that a task is not the factor of problems occurring during a test. For that assessment, a single-ease question is asked.

Group:	Scenario 1 - Navigating				Scenario 2 - Modelling			
	a)		b)		a)		b)	
	Editor	Questions	Editor	Questions	Editor	Model	Editor	Model
Group 1	Papyrus	Set 1	MS	Set 2	Papyrus	1	MS	2
Group 2	Modelling Studio (MS)	Set 1	Papyrus	Set 2	MS	1	Papyrus	2

Table 6.1.: Order of editors and questions for the different groups of the user test

The target group of the user-test are individuals with diverse level of understanding of UML or SysML modelling, from beginners to experts.

6.1.2. Tasks

The instructions of the tasks can be found in Appendix A.2 on page 75. There are two types of tasks, one for navigation and one for modelling. Tasks for navigation are instructions and questions for the user about the model that can only be answered by exploring and searching through the model. The modelling task instruct the user to replicate the given model.

6.1.3. Questionnaire

The full set of questions in the questionnaire can be found in Appendix A.2.3 on page 80. The general structure of the questions is as follows:

- Demographic data
- Editor-specific Questions: Papyrus
- Editor-specific Questions: Modelling Studio + Additional Questions for new elements
- Concluding questions
- Single-ease questions for tasks (asked after each task, just gets entered here)

The editor-specific questions have their own internal structure:

- AttrakDiff 1-3
- Hierarchical questions about visual importance, placement, and sizing of editor parts
- Modelling interactions questions
- Navigation interactions questions

AttrakDiff

The AttrakDiff questionnaire originates from a work model developed by Hassenzahl that showed through studies that rating of attractiveness consists of hedonic and pragmatic qualities (Hassenzahl 2006; Hassenzahl, Burmester, & Koller 2022). However, both are perceived independently of one another. Additionally, in AttrakDiff the Hedonic Quality (HQ) is split in the sub-qualities, stimulation (HQ-S) and identity (HQ-I). (Hassenzahl et al. 2022)
The attractiveness is measured using a range between two opposite adjectives, known as semantic differentials. The value range of the semantic differentials is from one to seven, with four as the neutral element. The differentials belong to one of these categories: Pragmatic (PQ), Hedonic Subquality Stimulation (HQ-S), Hedonic Subquality Identity (HQ-I) and Attractivenss (ATT). In total 28 word pairs are used, seven of them belong to each specific category.

6.1.4. Confounders

The following explains the existing confounders and how they are addressed.
The first confounder is the different levels of knowledge of modelling theory. This is limited by a brief introduction into UML by the interviewer if necessary. Before the test, the person is asked if they understand the given model, if not fully a quick explanation can be done.
Secondly, the knowledge about modelling in Papyrus is beneficial in navigating the editor. The person's knowledge level is recorded and based on the knowledge level the participants can be grouped and compared.
Thirdly, investigating speed metrics of individuals navigating menus in both editors can distort the comparison because the prototype editor does not have the same feature set implemented. Therefore, speed is not used during evaluation. However, Modelling Studio has buttons in the main UI representing the same functionality as Papyrus and if a user interacts with such an element, it is communicated that that feature is not implemented. Elements that are not needed for modelling but are present by Eclipse by default are removed.
The forth confounder is that the model is known to the user after the first editor. The effect is negated by switching the order of editors between the test groups, as proposed in the user-test setup before.

6.1.5. Setup

Each test person was tested individually. The participant was seated at a table with the test setup in front of them. The research worker was sitting to their right and taking notes. The participant gets a printed list of tasks to accomplish. The research worker only interrupts if asked to help or an error outside the expected occurred.
The test setup consists of a monitor (27in diagonal), keyboard (English or German layout based on participant's preference), mouse (with dedicated mouse back and forward button) and the computer. An illustration of the setup can be seen in figure 6.1. The computer also records the screen.
During the test, the users were asked to "think out loud".

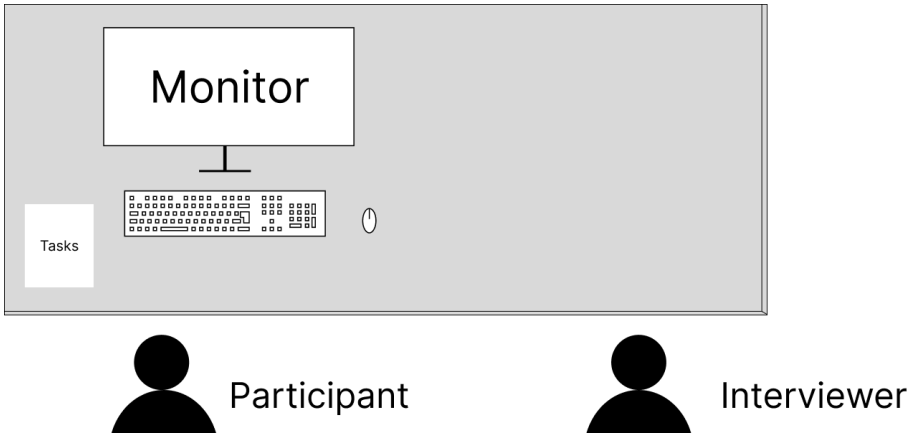


Figure 6.1.: Setup of Physical Test Environment (Source: Own Representation)

6.1.6. Processing

For processing, a Python (Guido van Rossum & Python Software Foundation 2022) Jupyter Notebook (Project Jupyter 2022) was used. Noteworthy libraries are:

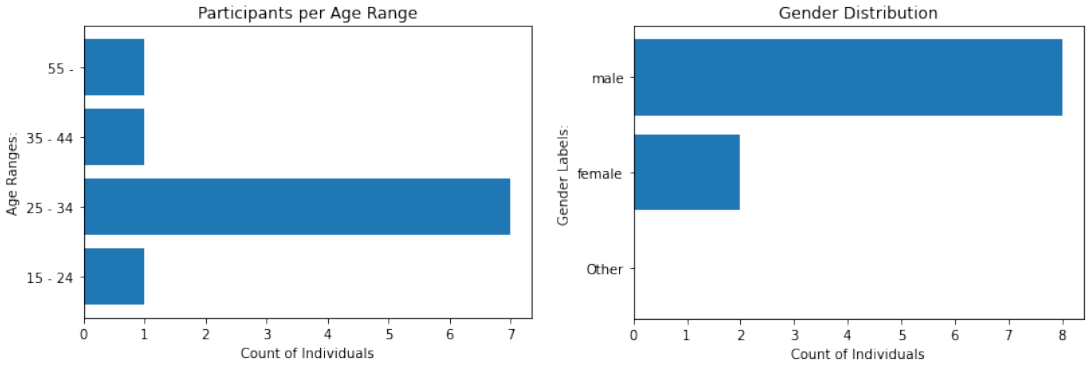
- Pandas for reading and handling data with data frames (Pandas-Dev 2022)
- Matplotlib for plotting (Matplotlib Developers 2022)
- SciPy for statistical tests (SciPy 2022a)

6.2. Results

The results are split into the parts: 6.2.1 Demographic, 6.2.2 Perceived Difficulty of Tasks, 6.2.3 AttrakDiff, 6.2.4 Hierarchy, 6.2.5 Aiding Diagram Composition, and 6.2.6 Navigation.

6.2.1. Demographic

In total, ten ($n = 10$) individuals participated in the usability study. Figure 6.2 shows the participant count per age group and gender. The gender distribution is as follows: 20% female, 80% male. The biggest age range of the participants is '25-34' with 7 participants (70%), the age ranges '15-24', '35-44', and '55-' have one participant (10%) each.



(a) Participants per Age Range (b) Participants per Gender

Figure 6.2.: Evaluation of Age Ranges and Gender in Demographic (Source: Own Representation)

Figure 6.3 shows how the participants categorized their knowledge level in UML or SysML before the test. For the editor Papyrus, four users had no prior experience (40%), three with beginner to intermediate expertise (30%) and three advanced to expert users (30%).

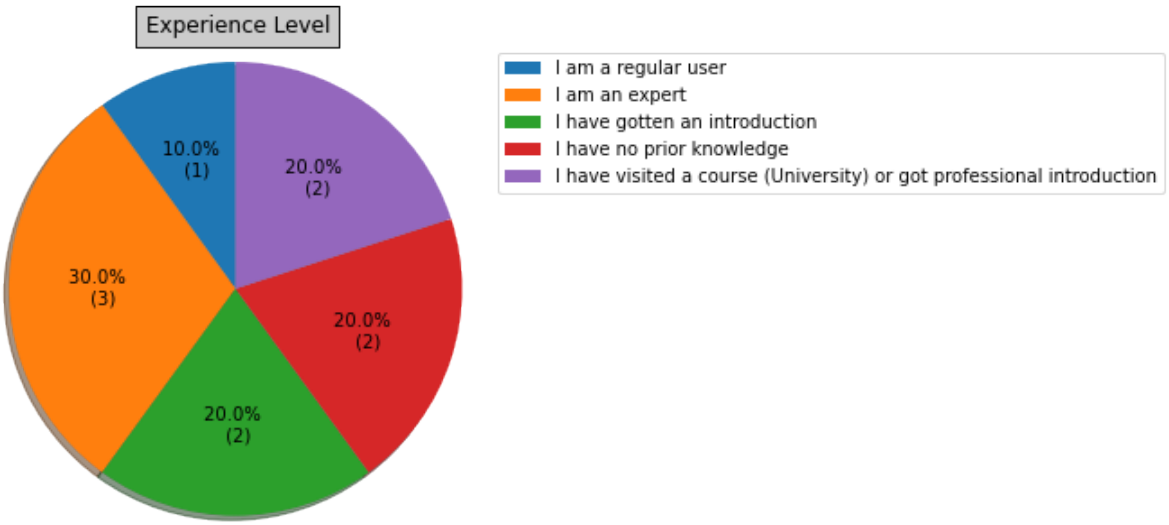


Figure 6.3.: Individual's Prior Experience with UML (Source: Own Representation)

6.2.2. Perceived Difficulty of Tasks

Table 6.2 shows the perceived difficulty of tasks. For the modelling tasks both have resulted in the same values, the average difficulty is 1.8 with a stand deviation of 0.789. Thus, the difficulty was perceived as similar. For the navigation tasks, the averages are not the same, but with a p-Value of 0.591 from the student's t-test for paired samples (SciPy 2022d), the hypothesis of identical averages of both data sets cannot be rejected.

	Navigation		Modelling	
	Questions 1	Questions 2	Model 1	Model 2
count	10	10	10	10
mean	1.600	1.700	1.800	1.800
std	0.699	0.823	0.789	0.789
min	1	1	1	1
max	3	3	3	3
t-statistic	-0.557		0	
p-value	0.591		1	

Table 6.2.: Results of Perceived Task Difficulty (Source: Own Representation)

6.2.3. AttrakDiff

For the purpose of AttrakDiff data processing, the differentials are shifted to the range of -3 to 3 and possibly flipped if the negative word was originally on the right side. For AttrakDiff, different plots can be created to view the results.

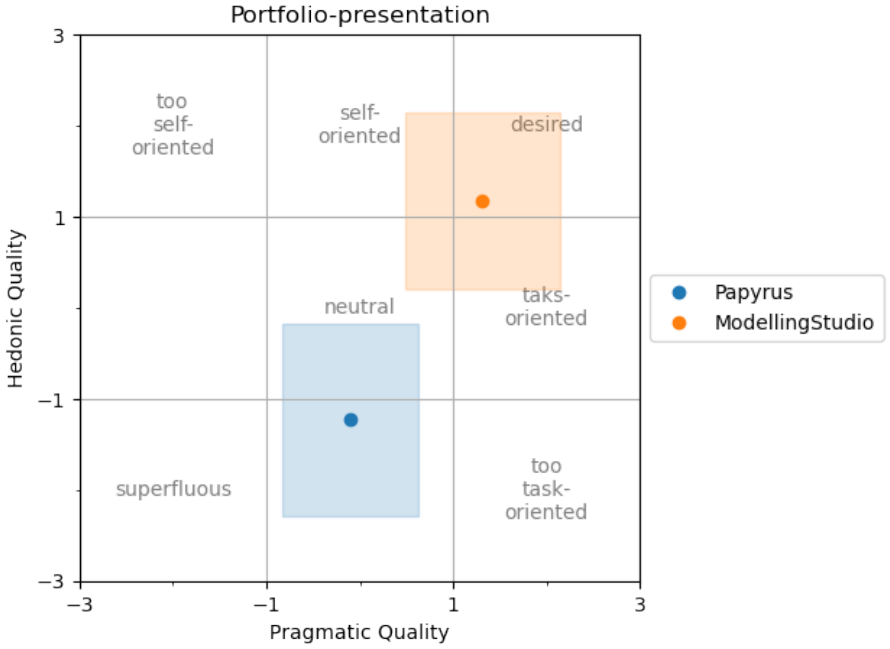


Figure 6.4.: AttrakDiff: Portfolios for Papyrus and Modelling Studio (Source: Own Representation)

The portfolios, figure 6.4, show the portfolio of both applications based on the means of the factors of the main categories of pragmatic (PQ) and hedonic (HQ) qualities. The area around a point represents the 95% interval in the category. Generally speaking, an application closer to the top-right is better perceived and a smaller area indicates a more precise result. Figure 6.5 show the means per category for both editors, for all categories modelling studio performed higher on average. The means for Modelling Studio compared to Papyrus are for PQ 1.31 to -0.10, for HQ-I 0.44 to -0.87, for HQ-S 0.72 to -0.35, and for ATT 1.31 to -0.71.

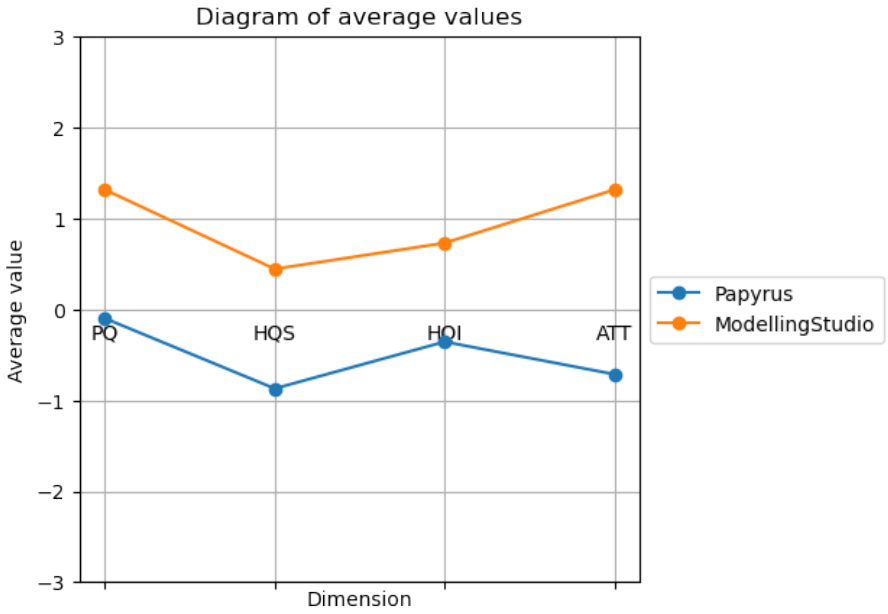


Figure 6.5.: AttrakDiff: Averages per Category for Papyrus and Modelling Studio (Source: Own Representation)

Figure 6.6 plots the 28 different semantic differentials. In the word pairs of "technical/human" and "undemanding/challenging" Papyrus outperformed Modelling Studio. For the pairs "cautious/bold", "unprofessional/professional", "unpresentable/presentable" the means are close, for the other 23 pairs Modelling Studio is rated higher than Papyrus.

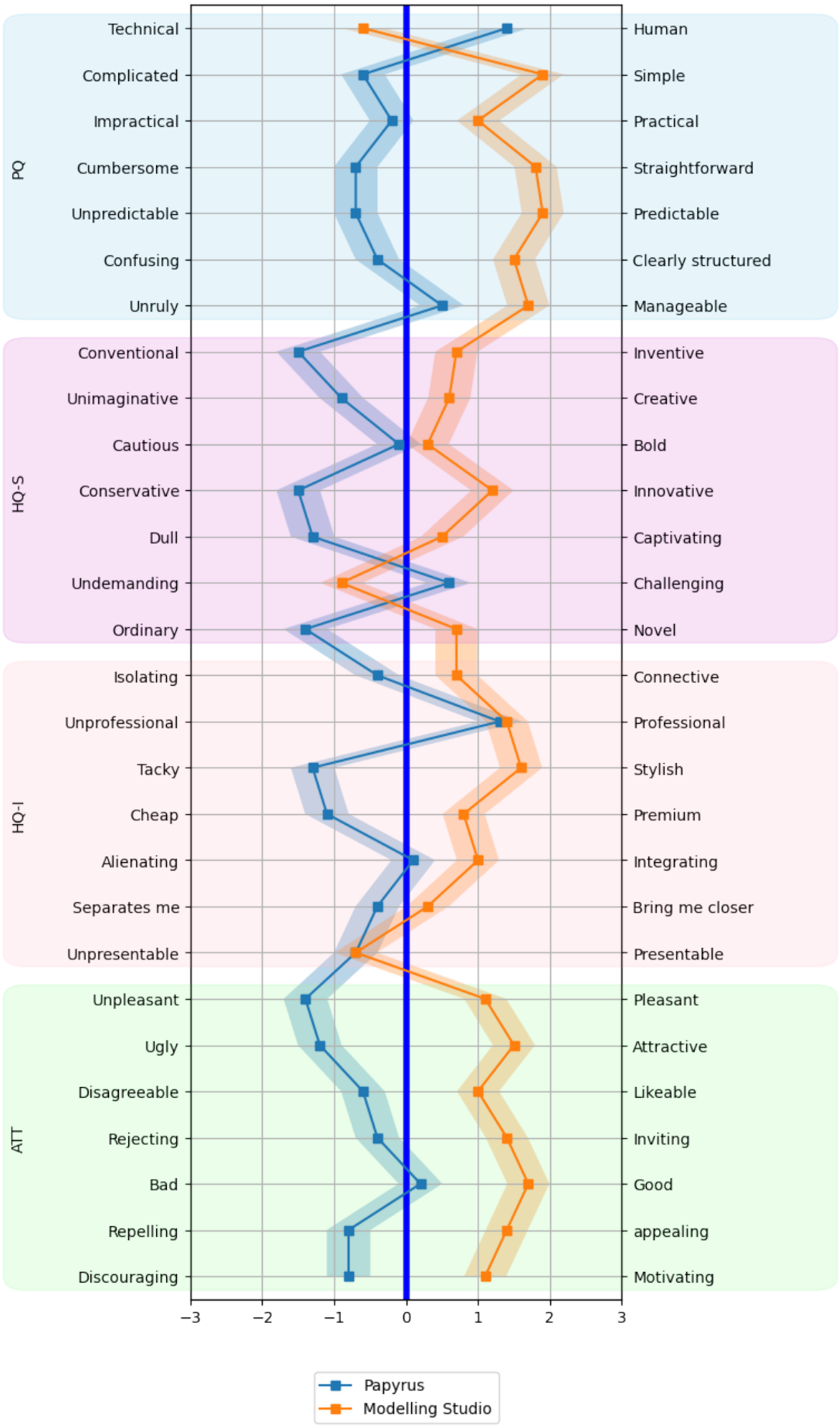


Figure 6.6.: AttrakDiff: Semantic Differentials for Papyrus and Modelling Studio (Source: Own Representation)

6.2.4. Hierarchy

The hypothesis for the hierarchy changes is: H_0 : Users are rating the changes in hierarchy in the parts of tab bar, property view, and palette higher in Modelling Studio than Papyrus, while the other parts show no decrease in rating of the element. To accept or reject the hypothesis, the placement ratings are analysed.
All placements use the student's t-test for related samples (SciPy 2022d) with a significance level of $\alpha = 0.05$, for tab-bar, property view and palette the one-sided test is applied to test if they are better; for others the two-sided test was used to determine if the underlying population is the same. The range of the placements questions is 1 ("bad") to 7 ("good").
For the file tree, toolbar and menubar the ratings are similar and the hypothesis that the samples are from the same population cannot be rejected.
The rating for the tab bar differs significantly, p-value of 0.011 (t-statistic: 2.72). Modelling Studio has a higher mean of 6.2 compared to 3.9 for Papyrus.
For the palette, the rating did improve in means from Papyrus 5.5 to 5.7 for Modelling Studio, but it does not indicate a significant change (t-statistic: 0.390, p-value: 0.352) so the hypothesis for the placement found an improvement has to be rejected.
Lastly, the property view showed improvements in the rating average from Papyrus, 4.5 to Modelling Studio 5.4. The difference is significant with a p-Value of 0.026 (t-statistic: 2.23).

Statistic	Editor	Palette	Tab bar	File Tree	Property View	Toolbar	Menubar
Mean	Papyrus	5.5	3.9	6	4.5	6	6.2
	M.Studio	5.7	6.2	6.3	6.2	5.4	6.3
std	Papyrus	1.71	2.37	1.24	2.06	1.24	1.22
	M.Studio	1.15	0.78	0.94	0.91	1.50	1.05
t-statistic		0.39	2.72	0.89	2.23	-1.15	0.20
p-value		0.352	0.011	0.393	0.026	0.278	0.840

Table 6.3.: Placement Ratings of hierarchical elements (Source: Own Representation)

Looking at the ranking of hierarchical parts, see table 6.4, for their visual importance one swap appears, between position 3 and 5. The property view and palette swapped places in the ranking, the property view is perceived with higher importance in Papyrus than in Modelling Studio. For the properties view in Papyrus, participant number 6 provided an explanation for their decision, stating that the property view is "messy".

Editor	1	2	3	4	5	6
Papyrus	Diagram Content	Diagram Area	Property View	File Tree	Palette	Menu and Toolbar
Modelling Studio			Palette		Property View	

Table 6.4.: Ranking of Hierarchical Parts following their perceived visual importance (Source: Own Representation)

6.2.5. Aiding Diagram Composition

The user can be aided in the diagram composition to create better diagrams by providing an alignment help, by changing default behaviour of arrows to follow best practices.

6.2.5.1. Diagram Alignment Helper

The most important part of the alignment helper for Modelling Studio are the Guidelines. The participants were asked to rate the guidelines from 1("bad") to 7("good") for Papyrus and Modelling Studio.
Modelling Studio has a higher mean of 6 (σ: 1.41) compared to Papyrus 4.9 (σ: 1.66). The rating for Modelling Studio is significantly better than Papyrus with a p-Value of 0.037 (t-statistic: 2.01, α: 0.05, SciPy (2022d)).
Addressing the numbers indicating spacing between elements, the participants found the numbers not very noticeable, relative predictable and are spread in opinion if they are helpful. Figure 6.7 shows the box plots for the ratings.

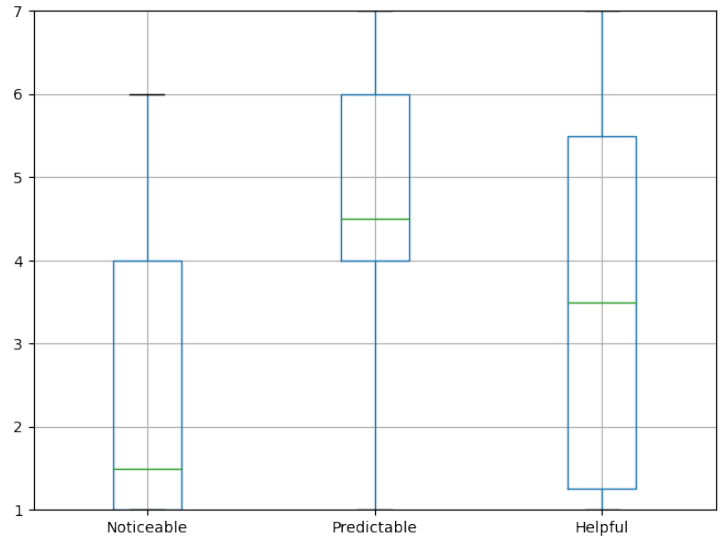


Figure 6.7.: Box Plots with Rating on how Noticeable, Predictable, Helpful Guideline Spacing Numbers are (Source: Own Representation)

Users were asked if they actively tried to align elements during the tasks. For Modelling Studio, 90% of users tried to actively align elements in the diagram, compared to 70% in Papyrus
An observation during the user test was that the users who started in Modelling Studio also tried to align items in Papyrus. While users starting in Papyrus did not always align items in the next editor. Splitting the users' claims that they actively aligned elements by the order of the editor results in two findings. Firstly, for the first editor that was tested, Papyrus was aligned 2 out of 5 times while Modelling Studio was aligned 5 out of 5 times. Secondly, for the second editor, Modelling Studio was aligned 4 out of 5 times and Papyrus 5 out of 5 times.

6.2.5.2. Default Arrows

The changes to the default arrows are rated by the user on a 1("bad") to 7("good") range.

The default arrows in Modelling Studio are rated significantly better than in Papyrus (t-statistic: 5.40, p-Value: 0.0002, α: 0.05, SciPy (2022d)). Modelling Studio has a mean of 6.0 (σ: 1.05) and Papyrus 2.8 (σ: 1.61).
Additionally, the usefulness of additional label information for composite and generalization arrow types are ranked. Both ratings have the same result, the mean for Modelling Studio is 5.9 (σ: 1.44) and for Papyrus 4.3 (σ: 2.16). Modelling Studio is significantly better with a p-value of 0.036 (t-statistic: 2.02, α: 0.05, SciPy (2022d)).
Participant 1 stated they "really liked [the] auto-rectify of the arrows" (P01) and P09 expressed "[n]ice arrow actions" for Modelling Studio.
An expert in Papyrus, P01, was the only person who knew / found the arrow option to make arrows rectangular in Papyrus.

6.2.5.3. Rating of Quick Creation Pop-up

The quick creation pop-up rating is presented in table 6.5. Modelling Studio has a higher mean than Papyrus of 6.1 to 4.6 and a lower deviation (0.99 to 2.22). Assuming, the hypothesis that the quick creation pop-up is better in Modelling Studio has to be rejected by the student's t-test for paired samples (SciPy 2022d) with a greater alternative. The p-Value of 0.052 is larger than the threshold of 0.05.

Quick Creation Pop-up		
	Papyrus	Modelling Studio
count	10	10
mean	4.600	6.100
std	2.221	0.994
min	1	4
max	7	7
t-statistic	1.799	
p-value	0.0526	

Table 6.5.: Rating of Quick Creation Pop-up (Source: Own Representation)

However, the subjective question which version the user prefers between Papyrus and Modelling Studio produced a p-Value of 0.0019 in a binominal test (SciPy 2022b) (k = 10, n = 10).
Furthermore, the users provided feedback for Papyrus that "Popup for new Property takes too long" (P01), "Pop-up of the properties for a block unwieldy" (P02, translated from German), "Quick creation popup appears late" (P03) and "Quick creation popup too slow" (P04).

6.2.6. Navigation

The hypothesis, introduced in 4.4, to improve the usability for navigating a model is composed of the following changes: Firstly, applying an icon to a valid link on an element, the link becomes more visible compared to Papyrus. Secondly, that removing the double click interaction on elements for link interaction causes the user to be less frustrated Furthermore, applying an icon to a valid link on an element, the link becomes more accepted compared to Papyrus.

Lastly, breadcrumbs provide a secondary navigation that is added that is not distracting and is predictable in navigation

6.2.6.1. Link Visibility

The hypothesis that the link visibility is better (greater) in Modelling Studio than Papyrus can be accepted. The populations with averages of Modelling Studio, 6.1 and Papyrus, 2.4 produce a p-Value of 9.989e-05 (t-statistic: 6.010), which is significant as it is below the significance level of $\alpha = 0.05$ (SciPy 2022d).

	Papyrus	Modelling Studio
count	10	10
mean	2.400	6.100
std	1.264	1.852
min	1	1
max	5	7
t-statistic	5.2152	
p-value	4.344e-05	

Table 6.6.: Results of Link Visibility Question (Source: Own Representation)

6.2.6.2. Navigating a Link

From observation during the tasks, 3 out of the 10 users had the "Hyperlink Creation Pop-up" (see figure 3.12) to appear in Papyrus at least once by accidental double clicks. The reaction received from "thinking out loud" was the users are more surprised than frustrated by the pop-up. The change of removing the double click on the whole element in Modelling Studio caused no major interruptions or calls for help by the user. All users who had advance to expert experience in Papyrus tried double-clicking the class like they are used to in Papyrus, but the transition to clicking the link was managed without external help.

6.2.6.3. Navigate Up Diagram Button / Comment

The comparison in rating for the navigate up button or comment implementation shows a significant difference (t-statistic=6.0, p-value=0.00010, SciPy (2022d)). The means are 2.7 for Papyrus and 6.3 for Modelling Studio, on a range from 1 ("bad") to 7 ("good").

6.2.6.4. Breadcrumbs

The hypothesis that the breadcrumbs are not distracting can be assessed by the ranking of visual importance from the questionnaire and observations. It can be accepted because the questionnaire question produced a p-Value of $2.95e-09$ (t-statistic=21.0, expected distribution mean=4, $\alpha = 0.05$) in a singe-sample t-test (SciPy 2022c). The mean is 6.8 with a min of 6 and a max of 7 on a range of 1 to 7 for the rating. Additionally, from observation during user tests, individuals had to return to the editor to recognize the breadcrumbs when answering the questions, as they were not noticed during the tasks.

The rating of the breadcrumbs produced a significant result as well (t-statistic=2.32, p-Value= 0.022, $\alpha = 0.05$, SciPy (2022c)), the mean 5.3 on a range from 1 ("bad") to 7 ("good") with a min of 2 and max of 7.
As the breadcrumbs can be used for navigation, the predictability was rated an average of 5.4 on a range from 1 ("unpredictable") to 7 ("predictable"). The predictability rating is also significant with a p-value of 0.003 (t-statistic=3.50, $\alpha = 0.05$, SciPy (2022c)).

7. Conclusion

At first, the results are discussed based on the hypothesis and state of the art, then the reflections about the implementation are made, and lastly an outlook is provided.

7.1. Discussion

The demography of participants offers diversity of experience levels and ages. The self categorization, see figure 6.3 for the knowledge in UML or SysML showed signs of individuals underestimating themselves. The people from the groups that visited a university course or received an introduction showed knowledge of a regular or expert user in the questions that were assessed by the practical tasks.
The prototype, Modelling Studio, accomplished usability improvements based on the increase in rating for most changes that were set out to be implemented. Generally, the usability has been improved, which was evaluated through the AttrakDiff questionnaire (6.2.3). The average scores for the different categories of AttrakDiff are on the positive side of the scale and based on the portfolio figure, 6.4 the trend towards the top right is positive. Modelling Studio performed better in all categories of AttrakDiff compared to Papyrus.
The tasks for modelling and navigation have no significant difference in their difficulty. The overall difficulty was leaning towards the "easy" side of the scale, with means between 1.6 and 1.8. Therefore, the confounder that the tasks are too difficult and thus affect the rating of the editor did not occur.

Hierarchy

Editor parts that should not be affected by the changes showed no significant change in their rating.
The repositioning of the task bar to the top was a significant change to the better, from an average of 3.9 to 6.2. This finding follows the recommendation of Google (2022b); Jakob Nielsen (2016) to place the element at the above of the content it should display.
The change to the property view produced a significant change in the rating in favour of Modelling Studio. Averages increased from 4.5 in Papyrus to 6.2 in Modelling Studio. The vertical stacking of elements and labels, as suggested by Matteo Penzo (2006), produced in a better result, but further research in accurate measuring timings for task completion is recommended.
The palette showed minor improvement of 0.2 in means from Papyrus 5.5 which is no significant difference. This means however that the palette was not negatively impacted by the swap with the property view.

Further research based on measuring task timings in create, read, edit, and delete tests, like Planas and Cabot (2020) used in their study, could provide additional information if the theoretical approach for optimizing the placements based on Fitts Law is applicable.

Aid in Diagram Composition

The alignment helpers guidelines are significantly better in Modelling Studio than Papyrus (Means: 6 to 4.9). However, the ratings for the numbers indicating the helpfulness with a mean of 3.6 and standard deviation 2.41 are dividing the users. Based on the mean of 4.6 in the predictability of behaviour, adjustments should be made.
The results from 6.2.5.2 about the ratings of default arrows show that applying the proposed best practice from Wong and Sun (2006) increased the rating compared to Papyrus from 2.8 to 6.0. The best practice of GC7"Draw arcs orthogonally" by Wong and Sun (2006) can be recommended.
The result of composite associations and generalization arrows in Modelling Studio (mean: 5.9, σ: 1.44) outperforming Papyrus (mean: 4.3, σ: 2.16) significantly shows that the criteria by Wong and Sun (2006), CC1 "Join inheritance arcs" and GC1: "Be selective" applied to the labels of associations are also preferred by the user.

Navigation

The visibility of a link could be improved from 2.4 to 6.1 in Modelling Studio. The change of Modelling Studio is significantly superior to Papyrus, which follows the basic UI design rule that an interaction should be visible and not hidden, see "Recognition rather than recall" by Nielsen and Molich (1990).
The fact that only users who had prior experience in Papyrus were expecting the double click functionality shows that for them the learning process of using another editor is stronger than the intuitive design. Raskin stresses that a learned procedure does not lay ground to call an UI intuitive, rather the opposite. If it has to be learned, it is not intuitive. (Raskin 1994)

Conclusions for modelling domain

A takeaway for the modelling domain is adhering to the criteria described by Wong and Sun (2006) also helps users in graphical editors, especially if the criteria are enforced without additional effort for the user.

7.2. Reflection

The prototype tested a wide range of changes that could be further improved through iterations based on the current implementation and findings. A limitation of the usability is that only a subset of SysML is supported.
In general, the prototype has more possibilities to incorporate accelerators and customization. The first customization that could provide help for a user is custom palette layouts. This allows

the user to place specific UML elements in the palette at a specific spot. Furthermore, the palettes could be customizable per diagram type.
The properties view in Modelling Studio allowed the user to create multiple lines of names for an element, which was confusing for some participants that wanted to press "Enter" to submit the changes. Modelling Studio does not require the user to submit the change, as it is done automatically. This behaviour should be altered to prevent confusion, a new line could still be created by using the common shortcut of "CTRL" and "Enter".
Also connected to the name field of the properties view, the cursor should automatically jump into the property view without the user having to click again. The behaviour was implemented, but it was working unreliably and could not be fixed in-time; thus it was removed for the user test.
The usage of the command pattern increased initial implementation effort, but turned out to be a good return on investment in later development. The prototype uses only a single stack for undo / redos instead of a multi-stack approach that is common for IDEs. A single stack was easier to implement, and the prototype did not require multiple stacks for the user test. However, a product should use a multi-stack approach if multiple diagrams can be edited at once.
The implementation for arrows is lacking in some situation which create not ideal paths. Arrows are hard to get right, Modelling Studio is also missing the functionality to move the arrow manually, as it only needed the auto-layout for the user-test.

7.3. Outlook

The developed prototype is in its infantry and can be improved based on the findings of this thesis and requires more iterations of improvement. Additionally, separate elements of the editor like the status indication for long-running task, navigation in context menus and usability in different screen sizes should be evaluated. Furthermore, Planas and Cabot asked for more opinionated decisions in editors while a few were addressed in this thesis, this could be expanded upon. From the user test, more information based on the movement of the mouse could be extracted to make observations based on Fitts law.
Addressing Papyrus a few of the changes can be activated, if they already exist, or implemented to improve the usability of the editor. Some of the changes are hide role and actor labels by default, change arrow creation behaviour to connect from the edges of an element, make the "rectify" style for arrows more accessible by advertising it or making it the default, and lastly optimization of the palette.

A. Appendix

A.1. Informed Consent

Informed consent for the "Modelling Studio" study

What is it about?

As part of the master thesis, an application was implemented that allows to view and model UML-based diagrams. The application in cooperates different paradigms found in user experience and modelling literature. The created application is compared to an existing editor used in the modelling domain named Eclipse Papyrus. Questions about to be answered in this study are connected to the paradigms that were implemented and basic user-experience evaluation.
To be able to answer some of these questions and to improve the prototype, your help is needed. Your steps to help are as follows:

1. Reading and signing this consent form for the processing of your data.
2. Briefing on the test procedure.
3. Carrying out the test.
4. Filling out a questionnaire.
5. Questions and comments on the test procedure.

What happens with your data?

Your name or other personal data are not included in any test or in the programme. We are not interested in your name or any other personal data. Your name only appears on this document in the form of your signature, and therefore cannot be traced back to a data record. In the questionnaire, however, you have to state your age range and gender. These are only recorded for statistical purposes so that diversity can be checked.

Risks

Due to possible rapid image changes in the software during use, flashes can occur, which may affect persons with photosensitive epilepsy. The test personnel are trained for such an eventuality and will immediately make an emergency call.

1

Contact

The following persons are responsible for the implementation of the software and this evaluation, and are also open and available at any time for questions regarding this evaluation:
— Janik Mayr [janik.mayr@students.fhv.at]

Declaration of consent

1. I have received sufficient information about this study, had the opportunity to ask questions and understand the contents.
2. I am aware that participation is voluntary and that I can withdraw from the study at any time. I do not need to provide a reason, and withdrawing does not have negative consequences for me.
3. I am aware that I can contact the person providing the information at any time if I wish to withdraw from the study prematurely. This person will ensure that the original state is restored (data deleted).
4. I know that all my data is subject to confidentiality and will be stored inaccessible to third parties.
5. All my data will be stored and processed pseudonymously so that it is not possible to identify me personally (e.g., P01 instead of my name).
6. I acknowledge that the results of the study may be published and used for future research purposes without having to ask for my consent again.
7. I have had sufficient time to make my decision.
8. I can ask for a copy of this informed consent form and an information sheet for participants in this study.

I have personally read this consent form and agree to participate in the study.

Parts:

☐ Editing
☐ Viewing / Navigation

Name (first, last): _____

Place, Date: _____

Signature: _____

2

A.2. Question and Instruction Sets

A.2.1. Instruction Sheet - Example Group 1

For group 2 only the order of editor names under a) and b) of point 2 and 3 is swapped.

Tasks - Group 1

Please follow the instructions provided below. If you have questions or are stuck or require help please let the interviewer know.

1. Demographic Info - Questionnaire

   Please fill in the questionnaire section about demographic and prior knowledge.
   The form will let you know to switch back to this task sheet.

2. Test for Navigation in a model

   In this step you will do a practical test for navigating a model inside an editor.
   For this task you will be handed additional sheets with instructions and questions.
   These questions can be answered by telling or showing answers in the editor to the interviewer.

   Example:
   Instruction: “Navigate to diagram with the name ‘x’”
   Your answer: “I am now on the diagram x”

   In general, please let the interviewer know what your thoughts are during the exercise (thinking-out-loud).

   You will be doing the editors in the order:

   a. Papyrus with Questions 1
   b. Modelling Studio with Questions 2

3. Test for Modelling

   This step is a practical test for modelling a UML diagram.
   You will try to model the given reference model.
   Each test is limited to 10 minutes, this limit can be reached quickly so do not rush.
   Try to achieve quality over quantity.

   In general, please let the interviewer know what your thoughts are during the exercise (thinking-out-loud).
   You will be doing the editors in the order:

   a. Papyrus with reference model 1
   b. Modelling Studio with reference model 2

4. Finish Questionnaire

   Please finish answering the questionnaire

A.2.2. Tasks

A.2.2.1. Navigation / Viewing - Questions 1

Questions & Instructions 1

Prerequisite:
Are the terms “file tree”, “properties view”, “tabs” familiar ?
Is the editor open ?

1. **Open the file tree**, if not open yet

2. **Navigate to the diagram 'PC_ConfigurationModel'.**
   Hint: It should be close to the root of the model

3. **Open the link of the 'Inputs' folder**

4. **Are you in the 'Inputs' diagram?**

5. **What is the type of the 'Input' block's property max price**

6. **Navigate back to the last diagram 'PC_ConfigurationModel'**

7. **Navigate to 'Product Architecture' diagram**

8. **What is the value of the 'hdcapacity' property of the block 'Application'**

9. **What is the value 'read-only' flag on the 'hdcapacity' property of the block 'Application'**
   Hint: The information can be viewed in the Properties View

10. **Navigate to the diagram 'Model Libraries'**

11. **How many specialisations are visible in the diagram for the general block 'Screen'**

12. **Close all tabs**

A.2.2.2. Navigation / Viewing - Questions 2

Questions & Instructions 2

Prerequisite:
Are the terms “file tree”, “properties view”, “tabs” familiar ?
Is the editor open ?

1. **Open the file tree**, if not open yet

2. **Navigate to the diagram 'PC_ConfigurationModel'.**
   Hint: It should be close to the root of the model

3. **Open the link of the 'Model Libraries' folder**

4. **Are you in the 'Model Libraries' diagram?**

5. **What is the value of the property 'efficiency' of each of the visible specialisations of the block ‘MB’**

6. **What is the datatype of the 'clockrate' property of the ‘CPU’ block**

7. **Navigate back to the last diagram 'PC_ConfigurationModel'**

8. **Navigate to 'Product Architecture' diagram**

9. **What is the range of cpus that can fit on a motherboard (motherboard => Block: ‘MB’)**
   Hint:Look at the association between block MB and block CPU

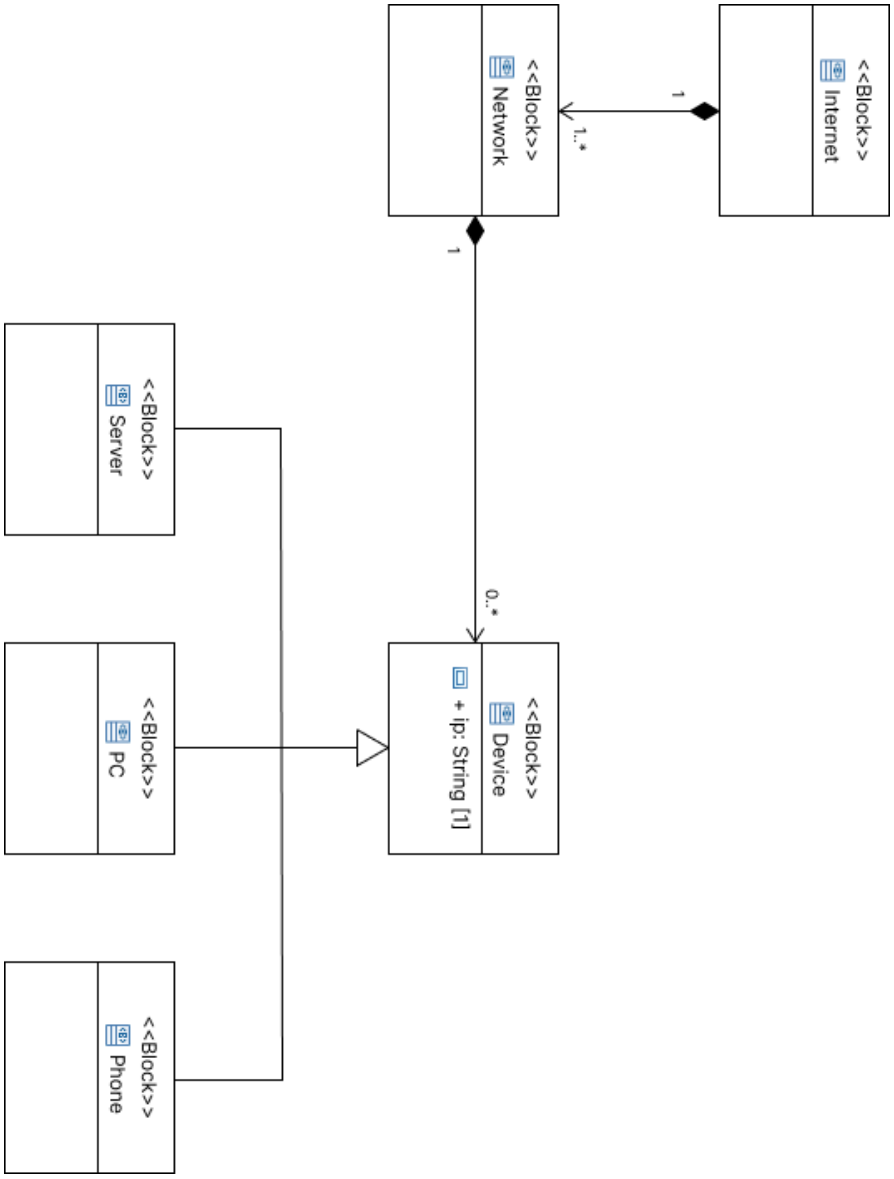
10. **What is the visibility of the ’HDUnit’ block’s property ’price’**

11. **Close all tabs**

A.2.2.3. Modelling – Reference Model 1

Reference Model 1

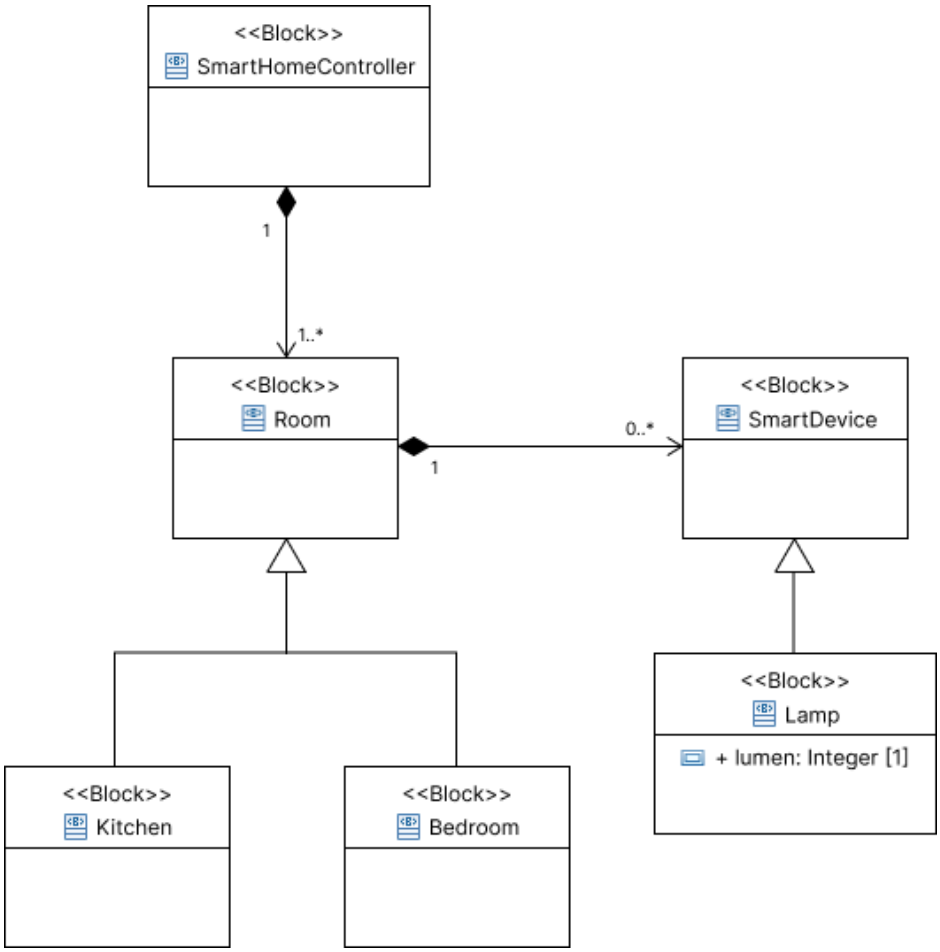
1. **Is the editor open?**
2. **Create a 'Block Definition Diagram' in the package 'Use Cases' with the name "usertask"**
3. **Recreate the following diagram, when done let the interview know**



A.2.2.4. Modelling – Reference Model 2

Reference Model 2

1. **Is the editor open?**
2. **Create a 'Block Diagram Definition' in the package 'Use Cases' with the name "usertask"**
3. **Recreate the following diagram, when done let the interview know.**



A.2.3. List of questionnaire questions

- Person Number
- Group Number
- What age range do you fall into?
- Which gender do you feel you belong to?
- What is your experience level in UML or SysML?
- Have you used UML or SysML in a professional environment?
- What experience do you have with the following editors?
  Multiselect for:
  - Eclipse Papyrus
  - MagicDraw
  - DrawIO
  - Visio
  - Visual Paradigm
  - StarUML
  - JetUML
- Is an editor you have used missing? Please tell which one?
- Shared Editor-Specific Questions
  For each editor, Papyrus first, Modelling Studio second.
  - AttrakDiff Questions see below at A.2.4
  - Rank the visual importance of the following elements for you personally?
    Ranking for (1 answer per column / row):
    - Menubar and Toolbar
    - Diagram Area
    - Palette
    - File Tree
    - Properties View
    - Diagram Itself
  - Is a important part missing for you?
  - How do you rate the Placement of the PALETTE in the editor?
  - How do you rate the Size of the PALETTE in the editor?
  - How do you rate the Placement of the TAB-BAR in the editor?
  - How do you rate the Size of the TAB-BAR in the editor?

  - How do you rate the Placement of the FILE-TREE in the editor?
  - How do you rate the Size of the FILE-TREE in the editor?
  - How do you rate the Placement of the PROPERTY-View in the editor?
  - How do you rate the Size of the PROPERTY-VIEW in the editor?
  - How do you rate the Placement of the TOOL-BAR in the editor?
  - How do you rate the Size of the TOOL-BAR in the editor?
  - How do you rate the Placement of the MENU-BAR in the editor?
  - How do you rate the Size of the MENU-BAR in the editor?
  - Remarks regarding any of the hierarchical elements?
  - Did you actively try to align the elements during the task?
  - How did the guide lines help aligning items?
  - How predictable is the creation of new elements?
  - Quick Creation Popup
  - How do you rate the default arrows that are created?
  - How do you rate usefulness of additional labels for default composite arrows?
  - How do you rate usefulness of additional labels for default generalization arrows?
  - What would you like to have different about the default arrows?
  - Remarks regarding modelling interactions?
  - How do you rate your awareness of which diagram you were viewing of the model?
  - Where did you look at to see what diagram you are currently in?
  - How do you rate the visibility of a Link on an element?
  - How do you rate the 'Back to x Diagram' Comment in the Diagrams?
  - Remarks for navigating and viewing a diagram?

- Specific Questions for Modelling Studio
  - How noticeable are the numbers indicating spacing between elements?
  - How predictable are the numbers indicating spacing between elements?
  - How helpful are the numbers indicating spacing between elements?
  - Which version do you prefer of Quick Creation Popup?
  - How do you rate the dedicated "Viewing mode"?
  - How do you rate the breadcrumbs element in the toolbar?
  - How predictable was navigating with the breadcrumbs?

- What program did you prefer for the task of Editing
- What program did you prefer for the task of Viewing / Navigating

- Any remarks?
- Task Difficulty (Asked during interview, entered now for easier evaluation)
  - Navigation - Questions 1
  - Navigation - Questions 2
  - Modelling - Questions 1
  - Modelling - Questions 2

A.2.4. AttrakDiff Questions

1. human/technical
2. isolating/connective
3. pleasant/unpleasant
4. inventive/conventional
5. simple/complicated
6. professional/unprofessional
7. ugly/attractive
8. practical/impractical
9. likeable/disagreeable
10. cumbersome/straightforward
11. stylish/tacky
12. predictable/unpredictable
13. cheap/premium
14. alienating/integrating
15. 'brings me closer to people'/ 'separates me from people'
16. unpresentable/presentable
17. rejecting/inviting
18. unimaginative/creative
19. good/bad
20. confusing/'clearly structured'
21. repelling/appealing
22. bold/cautious
23. innovative/conservative
24. dull/captivating
25. undemanding/challenging
26. motivating/discouraging
27. novel/ordinary
28. unruly/manageable

References

Android Developers. (2019, December). *Kotlin Overview.* Retrieved 2022-03-22, from https://developer.android.com/kotlin/overview

Android Developers. (2022). *Android Compose Tutorial | Android Developers.* Retrieved 2022-03-23, from https://developer.android.com/jetpack/compose/tutorial

Anna Kaley. (2019, March). *Contextual Menus: Delivering Relevant Tools for Tasks.* Retrieved 2022-05-13, from https://www.nngroup.com/articles/contextual-menus/

Arrow. (2021, December). *Arrow.* Retrieved 2022-07-04, from https://github.com/arrow-kt/arrow

Aurora Harley. (2019, December). *Accelerators Allow Experts to Increase Efficiency.* Retrieved 2022-03-29, from https://www.nngroup.com/articles/ui-accelerators/

Bajaj, M., Zwemer, D., Yntema, R., Phung, A., Kumar, A., Dwivedi, A., & Waikar, M. (2016, July). MBSE++ - Foundations for Extended Model-Based Systems Engineering Across System Lifecycle. *INCOSE International Symposium*, *26*(1), 2429–2445. Retrieved 2022-06-30, from https://onlinelibrary.wiley.com/doi/10.1002/j.2334-5837.2016.00304.x doi: 10.1002/j.2334-5837.2016.00304.x

Benjafield, J. G. (1992). *Cognition.* Englewood Cliffs, N.J.: Prentice Hall. (OCLC: 23648769)

Charness, G., Gneezy, U., & Kuhn, M. A. (2012, January). Experimental methods: Between-subject and within-subject design. *Journal of Economic Behavior & Organization*, *81*(1), 1–8. Retrieved 2022-07-03, from https://linkinghub.elsevier.com/retrieve/pii/S0167268111002289 doi: 10.1016/j.jebo.2011.08.009

Eclipse Foundation. (2021, August). *EMF: Eclipse Modeling Framework.* Retrieved from git.eclipse.org/c/emf/org.eclipse.emf.git/

Eclipse Foundation. (2022, May). *EMF Cloud - Model Server.* eclipse-emfcloud. Retrieved 2022-07-04, from https://github.com/eclipse-emfcloud/emfcloud-modelserver (original-date: 2019-11-01T20:35:55Z)

Eichelberger, H. (2003). Nice class diagrams admit good design? In *Proceedings of the 2003 ACM symposium on Software visualization - SoftVis '03* (p. 159). San Diego, California: ACM Press. Retrieved 2022-06-30, from http://portal.acm.org/citation.cfm?doid=774833.774857 doi: 10.1145/774833.774857

Erich Gamma, Richard Helm, Ralph Johnson, & John Vlissides. (1995). *Design patterns: elements of reusable object-oriented software.* Reading, Mass: Addison-Wesley.

Fitts, P. M. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of experimental psychology*, *47*(6), 381. (Publisher: American Psychological Association)

Gibson, J. J. (1979). *The ecological approach to visual perception: classic edition.* Psychology Press.

Goldstein, E. B. (2010). *Sensation and perception* (8th ed ed.). Belmont, CA: Wadsworth, Cengage Learning.

Google. (2022a). *Material Design.* Retrieved 2022-03-05, from https://material.io/design

Google. (2022b). *Tabs - Material Design.* Retrieved 2022-07-04, from https://material.io/components/tabs

Google, & Square. (2022, February). *Dagger.* Google. Retrieved 2022-07-04, from https://github.com/google/dagger (original-date: 2013-02-01T23:14:14Z)

Gosling, J., & Sun Microsystems. (2021, March). *Java.*

Graham, E. D., & MacKenzie, C. L. (1996). Physical versus virtual pointing. In *Proceedings of the SIGCHI conference on Human factors in computing systems common ground - CHI '96* (pp. 292–299). Vancouver, British Columbia, Canada: ACM Press. Retrieved 2022-05-30, from http://portal.acm.org/citation.cfm?doid=238386.238532 doi: 10.1145/238386.238532

Guido van Rossum, & Python Software Foundation. (2022, June). *Python.* Retrieved 2022-07-08, from www.python.org

Hale, K. (2007, October). *Visualizing Fitts's Law.* Retrieved 2022-04-17, from http://www.particletree.com/features/visualizing-fittss-law/

Hans Dockter, Adam Murdoch, Szczepan Faber, Peter Niederwieser, Luke Daley, Rene Gröschke, & Daz DeBoer. (2022, February). *Gradle.* Retrieved 2022-07-04, from https://github.com/gradle/gradle/releases/tag/v7.4.0

Hassenzahl, M. (2006). Hedonic, Emotional, and Experiential Perspectives on Product Quality. In C. Ghaoui (Ed.), *Encyclopedia of Human Computer Interaction* (pp. 266–272). IGI Global. Retrieved 2022-07-03, from http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-59140-562-7.ch042 doi: 10.4018/978-1-59140-562-7.ch042

Hassenzahl, M., Burmester, M., & Koller, F. (2022). *AttrakDiff.* Retrieved 2022-07-03, from http://www.attrakdiff.de/sience-en.html#publikationen

International Organization for Standardization [ISO]. (2018). 9241-11 (2018) Ergonomics of human-system interaction—part 11: usability: definitions and concepts. *International Organization for Standardization. https://www. iso. org/obp/ui/# iso: std: iso*, *9241*(11).

Ivanov, A. (2022, July). *Decompose.* Retrieved 2022-07-04, from https://github.com/arkivanov/Decompose (original-date: 2021-12-10T14:56:48Z)

Jakob Nielsen. (2016, July). Tabs, Used Right. *Nielsen Norman Group*. Retrieved 2022-07-04, from https://www.nngroup.com/articles/tabs-used-right/

Jakob Nielsen. (2020, March). 10 Usability Heuristics for User Interface Design. *Niels Norman Group*. Retrieved 2022-03-12, from https://www.nngroup.com/articles/ten-usability-heuristics/

JetBrains. (2022). *Compose Multiplatform.* Retrieved 2022-07-04, from https://github.com/JetBrains/compose-jb

JetBrains, & Open-source Contributors. (2022, March). *Kotlin.* JetBrains. Retrieved 2022-03-22, from https://kotlinlang.org/

Knight, C., & Munro, M. (1999). Comprehension with[in] virtual environment visualisations.

In *Proceedings Seventh International Workshop on Program Comprehension* (pp. 4–11). Pittsburgh, PA, USA: IEEE Comput. Soc. Retrieved 2022-07-03, from http://ieeexplore.ieee.org/document/777733/ doi: 10.1109/WPC.1999.777733

Laubheimer, P. (2018, December). Breadcrumbs: 11 Design Guidelines for Desktop and Mobile. *Nielsen Norman Group*. Retrieved 2022-05-10, from https://www.nngroup.com/articles/breadcrumbs/

Marr, D. (1982). *Vision: A computational investigation into the human representation and processing of visual information*. MIT press.

Matplotlib Developers. (2022, May). *Matplotlib*. Matplotlib Developers. Retrieved 2022-07-08, from https://github.com/matplotlib/matplotlib (original-date: 2011-02-19T03:17:12Z)

Matteo Penzo. (2006, July). *Label Placement in Forms*. Retrieved 2022-07-04, from https://www.uxmatters.com/mt/archives/2006/07/label-placement-in-forms.php

Moore, P., & Fitz, C. (1993). Gestalt theory and instructional design. *Journal of technical writing and communication*, *23*(2), 137–157. (Publisher: SAGE Publications Sage CA: Los Angeles, CA)

Nielsen, J. (1994). Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (pp. 152–158).

Nielsen, J. (2007, April). Breadcrumb Navigation Increasingly Useful. *Nielsen Norman Group*. Retrieved 2022-07-06, from https://www.nngroup.com/articles/breadcrumb-navigation-useful/

Nielsen, J., & Molich, R. (1990). Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 249–256).

Pandas-Dev. (2022). *pandas - Python Data Analysis Library*. Retrieved 2022-01-26, from https://pandas.pydata.org/

Petre, M., Blackwell, A., & Green, T. (1996, July). Cognitive Questions in Software Visualisation.

Planas, E., & Cabot, J. (2020, January). How are UML class diagrams built in practice? A usability study of two UML tools: Magicdraw and Papyrus. *Computer Standards & Interfaces*, *67*, 103363. Retrieved 2022-03-23, from https://linkinghub.elsevier.com/retrieve/pii/S0920548918303659 doi: 10.1016/j.csi.2019.103363

Project Jupyter. (2022). *Project Jupyter*. Retrieved 2022-01-26, from https://jupyter.org

Raluca Budiu. (2018, May). Between-Subjects vs. Within-Subjects Study Design. *Niels Norman Group*. Retrieved 2022-07-03, from https://www.nngroup.com/articles/between-within-subjects/

Raskin, J. (1994, September). Viewpoint: Intuitive equals familiar. *Communications of the ACM*, *37*(9), 17–18. Retrieved 2022-07-03, from https://dl.acm.org/doi/10.1145/182987.584629 doi: 10.1145/182987.584629

Rosenthal, R. (1976). *Experimenter effects in behavioral research*. IRVINGTON PUBIISHERS, lnc., New York. Retrieved 2022-07-03, from https://scholar.google.com/scholar_lookup?title=Experimenter%20Effects%20in%20Behavioral%20Research&publication_year=1976&author=R.%20Rosenthal

Saji. (2014, December). *Psychological Resources: Theories of perception.* Retrieved 2022-05-14, from http://psychologicalresources.blogspot.com/2014/12/theories-of-perception.html

SciPy. (2022a). *SciPy.* Retrieved 2022-01-26, from https://scipy.org/

SciPy. (2022b). *scipy.stats.binomtest — SciPy v1.8.1 Manual.* Retrieved 2022-07-05, from https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.binomtest.html#scipy.stats.binomtest

SciPy. (2022c). *scipy.stats.ttest_1samp — SciPy v1.8.1 Manual.* Retrieved 2022-07-06, from https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_1samp.html#scipy.stats.ttest_1samp

SciPy. (2022d). *scipy.stats.ttest_rel — SciPy v1.8.1 Manual.* Retrieved 2022-07-04, from https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_rel.html

Solso, R. L. (1998). *Cognitive psychology.* Boston: Allyn and Bacon. (OCLC: 37606160)

StatCounter. (2022, July). *Desktop Screen Resolution Stats Worldwide.* Retrieved 2022-07-07, from https://gs.statcounter.com/screen-resolution-stats/desktop/worldwide

Sun, D., & Wong, K. (2004). On understanding software tool adoption using perceptual theories. In *4th International Workshop on Adoption-Centric Software Engineering* (p. 51). IET.

theapache64. (2021, May). *compose-desktop-template.* Retrieved 2022-07-04, from https://github.com/theapache64/compose-desktop-template

Tilley, S., & Huang, S. (2003). A qualitative assessment of the efficacy of UML diagrams as a form of graphical documentation in aiding program understanding. In *Proceedings of the 21st annual international conference on Documentation - SIGDOC '03* (p. 184). San Francisco, CA, USA: ACM Press. Retrieved 2022-07-03, from http://portal.acm.org/citation.cfm?doid=944868.944908 doi: 10.1145/944868.944908

Weilkiens, T. (2016). *Variant Modeling with SysML.* MBSE4U - Tim Weilkiens. (ISBN: 9783981787542 OCLC: 970014948)

Wong, K., & Sun, D. (2006, September). On evaluating the layout of UML diagrams for program comprehension. *Software Quality Journal*, *14*(3), 233–259. Retrieved 2022-05-13, from http://link.springer.com/10.1007/s11219-006-9218-2 doi: 10.1007/s11219-006-9218-2

Statement of Affirmation

I hereby declare that all parts of this thesis were exclusively prepared by me, without using resources other than those stated above. The thoughts taken directly or indirectly from external sources are appropriately annotated. This thesis or parts of it were not previously submitted to any other academic institution and have not yet been published.

Dornbirn, 10^{th} July 2022 Janik Mayr, BSc