VRIJE
UNIVERSITEIT
BRUSSEL

Graduation thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Applied Sciences and Engineering: Computer Science

ENABLING THE GENERATION OF USER INTERFACES FOR PERSONALISED IOT APPLICATIONS FOR NON-DESIGNERS

BRENDA ORDOÑEZ LUJAN

Academic year 2022–2023

Promoter:	Dr. Audrey Sanctorum,
	Prof. Dr. Beat Signer
Advisor:	Dr. Audrey Sanctorum

Faculty of Sciences and Bio-Engineering Sciences

Abstract

In the past, when someone wanted to control smart home devices in a single application, they had to belong to the same brand or platform. Nowadays, there are IoT applications that can integrate and control different devices from different brands in a single platform, and day by day this integration improved, incorporating even more devices. However, we have noted that some IoT applications do not let the user determine how the user interface looks, while other applications that do, are targeted at users who have UI design and/or programming experience.

Users should be allowed to make their own IoT user interface designs, as it has been established over the years that the impact of the interface design is the most important and critical to its later use, but as we mentioned above, current solutions do not consider users who do not have programming or UI design experience. To solve this problem, the eSPACE authoring tool along with its framework and conceptual model is presented. This tool is part of the current solutions for integrating and controlling smart devices and *things* on a single platform and enabling users to create their own user interface for their respective IoT applications according to their own needs and preferences. However, while the current option for creating applications with the eSPACE authoring tool allows users to build any user interface they want without programming, it still requires users to have UI design knowledge, as a lack of these skills results in certain poor design choices. This fact leads to a question on how can we provide a user interface generation module that allows end users to create their own IoT application UIs without any prior programming knowledge or UI design skills?

In this thesis we will present some related work in the field of automatic user interface generation, which will serve as a basis for formulating the requirements of our solution to address these problems along with the concepts needed to extend the tool. This result will be implemented and evaluated by our target user to confirm that our objectives have been met.

Acknowledgements

I am profoundly grateful to have had the support and guidance of numerous individuals throughout this journey, and I would like to take this opportunity to express my appreciation.

First and foremost, I express my deepest and sincere gratitude to my supervisor, Dr Audrey Sanctorum, for her time, dedication and insightful feedback. Her guidance has been crucial in determining the direction and quality of this work. I am also grateful to Prof. Dr. Beat Signer, my promoter, for their important contributions and expertise. His thoughtful observations and constructive comments have greatly enriched this project.

I extend my heartfelt thanks to my family for their boundless love, encouragement and sacrifices. Without them I would not have had the opportunity to step out of my comfort zone and find myself in this situation. Their constant faith in me and their unwavering support is what makes me complete my projects.

To my friends, thank you for being by my side through the ups and downs of this journey. A special thanks to my boyfriend, together we embarked on this journey, and his encouragement has been essential in keeping my sanity. To his family, who took me in from day one, their support in every aspect has allowed me to make my way through this beautiful country.

Finally, this thesis would not have been possible without the collective support of these remarkable individuals. Each of you has played an indispensable role in shaping both my academic and personal growth, and for that, I am truly thankful.

Contents

1 Introduction
1.1 Problem Statement 2
1.2 Research Questions and Objectives 2
1.3 Research Approach and Methodology 4
1.4 Thesis Outline 5

2 Related Work
2.1 User Interface Design 7
2.2 User Interface Design Tools 9
2.2.1 Automatic User Interface Generation 11
2.2.2 Semi-automatic UID tools 12
2.2.3 Fully Automatic UID tools 17
2.3 Summary and Conclusions Drawn from Related Work 27
2.4 Resulting Requirements 30

3 Background
3.1 The eSPACE Framework 33
3.1.1 The eSPACE App Creation Process 37
3.2 Guidelines and Heuristics 39

4 Design and Implementation
4.1 Generation of IoT Applications 43
4.1.1 App Generation Process 44
4.1.2 Front-end Design 46
4.1.3 Default IoT Application Design 46
4.2 Implementation 47
4.2.1 Technology 47
4.2.2 Generate App View 47
4.2.3 The Final User Interface 52
4.3 Use Case Demonstration 54
4.4 Summary and Discussion 59

5 Evaluation
5.1 Setup 65
5.2 Participants 66
5.3 Protocol 66
5.3.1 User Study Presentation 66
5.3.2 Tasks Execution 66
5.3.3 Questionnaire and Interview 67
5.4 Results 67
5.4.1 Questionnaires 68
5.4.2 Observations 71
5.5 Summary and Discussions 73

6 Conclusion
6.1 Conclusion 77
6.2 Future Work 80

A Appendix A

B Appendix B
B.1 Scenario 91
B.2 Task Document 93
B.3 Post-Survey Questionnaires 95
B.3.1 Personal Information 96
B.3.2 Post-Study System Usability Questionnaire (PSSUQ) . 97
B.3.3 Post-Study Informative Questionnaire 98

C Appendix C

1 Introduction

From its earliest days to the present day, the Internet of Things (IoT) remains relevant and continues to impact our daily life. It has revolutionised communication and enabled new forms of control over smart objects. As a result of this influence, new smart devices and *things* entered the market, requiring the development of platforms for controlling and integrating them, such as Smart Home applications and authoring tools. There are various IoT authoring tools that have been developed in this field, some more complete than others. Some of them provide no control over the user interface (UI) that the end user must use, while the remaining ones that do, are meant for end users with an understanding of UI design or who have programming skills. There is no solution that does not require training or prior knowledge to create UI designs for IoT applications intended for end users.

The UI is crucial for end users because a well-designed UI enhances the user satisfaction, usability, efficiency, increases productivity and reflects the success of a product. Since it is difficult to predict the needs and preferences of different users for the same application, the need to provide them with solutions to create and customise their own designs requires no justification. The closest solution to this lack of attention on end users without design skills in the context of IoT applications was found in graphical recommender systems, such as the recommendation system for the UI designer proposed by

Ali [1], which effectively help the end user to create better UIs fitted to their own preferences; however this solution takes a lot of time and effort from the end users to improve their IoT user interface after it has been created; not to mention that this solution does not integrate the functionality of the application yet, meaning that it also requires another manual effort to finish creating the IoT application. Because of this, it may cause frustration towards end users who wish to create their own IoT applications with minimal effort.

We therefore see a need to offer a solution that enables end users to generate their own IoT applications based on their requirements, needs, and preferences with as little effort as possible and without the need for programming knowledge or UI design skills.

1.1 Problem Statement

IoT keeps evolving as new devices enter the market. Developers are creating or extending solutions to manage all technologies in one place; however, the difficulty arises in designing the user interface. The effort devoted to the user interface is substantially different from developing the functionality of such solutions. Developers must take into account user needs, which are complex, divergent and often evolve over time. Since user interface design has a significant impact on the usability of applications, users should ideally be able to design their own user interfaces and programmes. Taking into account that not all users have the same skills as user interface designers, we propose a solution by offering them the possibility to generate their own IoT applications based on their preferences and needs.

1.2 Research Questions and Objectives

Main Research Question How can we provide a user interface (UI) generation module that allows end users to create their own Internet of Things (IoT) application UIs without any prior programming knowledge or UI design skills?

In order to structure our solution, we must create sub-questions in response to this main research question. First, we will investigate the requirements of a user interface design tool for an end user to be able to create their own user interface capable of controlling IoT devices. To do this, we will look into related work on generation of user interface design tools. This brings us to the first research question, which takes into account the fact that our target user has little or no knowledge in UI designs or programming skills.

Research Question 1 (RQ1) What are the necessary requirements for the end user to generate a user interface without programming or UI design skills?

In order to meet the requirements we will carry out an analysis of the related work in the field of automatic user interface generation tools. As a result, we will provide an overview of existing tools as well as their limitations, concepts and techniques that can be useful to justify our later contributions.

The derived requirements from this analysis will give us a better understanding of what we need in our module for the end user to generate UIs but we still need to think about how we can approach this for an IoT application that controls smart devices, which brings us to the next research question.

Research Question 2 (RQ2) What are the necessary concepts needed for creating a module that allows end users to generate their own UIs for controlling their smart devices and *things*?

Instead of starting from scratch we want to start from an existing solution that provides control over smart devices; however, as we will discuss later, it is not an intuitive task to extend solutions that support the requirements from RQ1. Therefore we must present the necessary concepts and components that will form the basis of our module, and to show that it satisfies the end user's needs, we need to build a prototype, which brings us to the last research question:

Research Question 3 (RQ3) How can we extend an existing UID tool with UI generation options as a result of the requirements and new concepts gained from RQ1 and RQ2?

The objective of this last research question is to create a proof-of-concept prototype of a UID tool capable of controlling smart devices and that meets the requirements and concepts discovered in RQ1 and RQ2. We will then conduct a user study to evaluate and analyse whether the tool has achieved our main goal and gain more insight to identify if adjustments or improvements are needed.

1.3 Research Approach and Methodology

We are adopting the Design Science Research Methodology (DSRM) for information systems [32] for the development of our research. This methodology consists of 6 sequential activities, which contribute to the structure of our thesis.

1. **Problem identification and motivation**
   The problem identified was described in Section 1.1, which identifies the need for end users to create the interface of their own applications, in this case IoT applications according to their needs and preferences. An activity that is currently assumed by the developer or provided with options that require prior training for their use. These current solutions are limited to providing a manual process to the end user, which can lead to new problems in the usability of the system.

2. **Define the objectives for a solution**
   The main objective of this thesis is to research and develop a module that can generate a user interface (UI) that enables users to develop their own IoT applications in accordance with their own requirements and demands without any prior programming or UI design knowledge. In order to answer and build a solution that meets our main objective, we break down this goal into three research questions which are developed in more detail in the previous section, see Section 1.2.

3. **Design and development**
   To achieve our research objectives, we have introduced a new module in the eSPACE authoring tool, designed specifically for generating IoT applications. Full details of the design and development processes for this module can be found in Chapter 4.

4. **Demonstration**
   Once the prototype was developed, we conducted a user study with a prepared scenario and tasks under a certain condition, the details are covered in Chapter 5. Although the DSRM proposes to include resources for demonstration of how to use the tool, this step was not considered during the user study since in our particular case we wanted to measure how easy and intuitive the tool is without any prior training. As a result, while completing the tasks for the user study, the participants experimented with the tool. However, later in Chapter 4 we will demonstrate how our approach works and the possibilities it offers in a use case.

5. **Evaluation**
   After the user study, we move on to the evaluation stage which is discussed in Chapter 5, although the evaluation is part of the methodology we are adopting, it was also done to satisfy one of our derived requirements described in Section 2.4. In which we are interested to see whether the prototype, as a UID tool, is of quality. For this, we designed a user study including post-survey questionnaires, among them the Post-Study System Usability Questionnaire (PSSUQ), which will help us to observe the metrics corresponding to the quality of the software. At the end of this activity, we were left with further improvements and considerations to address based on the findings and observations of the user study, which we give in more detail in Chapter 6 as conclusions and ideas for future work.

6. **Communication**
   The main contribution of our thesis is the introduction of a new module for generating IoT applications within the eSPACE authoring tool. This module has been developed by analysing technologies in the field of Automatic UI Generation and incorporating concepts for creating UI to control smart devices and *things*. Our methodology, design, implementation, evaluation and conclusions are included in this dissertation, which will be presented during the thesis defence in front of a jury.

1.4 Thesis Outline

This thesis contains six chapters. We start by explaining the general context of the thesis in Chapter 1, identifying the problem to be solved, defining our research questions, and introducing the methodology used for our research. In Chapter 2, we present a brief history of user interface design, as well as the categorisation of user interface design (UID) tools along with several related work in the context of automatic UID tools. Additionally, we discuss the techniques and mechanisms identified in the generation of UIs to understand the general process, and derive the necessary requirements based on this analysis that our module must satisfy. In Chapter 3, we introduce the main tool that we will use as a basis to generate IoT applications. We provide both the fundamental concepts and the technical components needed to model our process. In addition, we also present a set of guidelines and heuristics that are part of the requirements identified in the previous chapter. In Chapter 4, we present the design and implementation of our IoT application generation module, as well as a use case of this module. We further demonstrate how our module satisfied the requirements derived from Chapter 2, and finish with a discussion on the limitations of the module. In Chapter 5, we present the evaluation of our module, share the results from surveys and observations made during the user study, and identify areas where improvements can be made.

Finally, in Chapter 6, we provide a summary of our entire study by answering to our research sub-questions, which are derived from our main research question presented in Chapter 1. In addition, we end with the future work that is inspired by the findings and observations of the user study, as well as the limitations and issues raised during the implementation of the module.

2
Related Work

In this Chapter, we discuss the concept and development of user interface design, as well as several user interface design tools related to the area of automatic user interface design generation. We will focus on answering our initial research question, for which we then provide an overview before concluding with a discussion of the resulting requirements that emerged from our analysis of related work.

2.1 User Interface Design

User interface design was not always as crucial as it is now. In fact, early user interfaces were relatively simple with rather awkward textual inputs and outputs, aimed at a single audience, in this case, scientists and engineers [18]. But as computers became more powerful and with the advent of new input devices, the user interface design became richer and began to take on relevance, as did the user interface design tools.
Let us first look at the purpose of user interfaces and what user interface design is, before we look at how user interface design have evolved over the years.

Purpose of user interface:
The purpose is to facilitate communication between the user and the com-

puter by hiding the structure, techniques, relationships and sequential interactions involving hardware and software [18].

Definition User Interface design:
"The general activity of user interface development, which consists of these tasks undertaken in a partially parallel, partially serial, partially iterative sequence: plan, research, analyse, design, implement, evaluate, document, train, maintain and recycle/replace" [17].

At a conference in 2002, two well-known product designers, Bill Moggeridge and Bill Verplank claimed that they invented the concept of user interface design in the early 1980s. However, a year earlier, Alan Cooper, a well-known programmer best known for creating the Visual Basic language, claimed that he was the first to define the concept of user interface design in the 1980s [17]. While in both cases these authors were contributors to the development of the concept, it cannot be denied that the term was already known to the general public before the 1980s. As Aaron Marcus, a specialist in user interface design and information visualisation, rightly points out that he was aware about it in the late 1970s [17]. Additionally, Alan Kay, a pioneering scientist whose work established the field of interactive computer graphics, explained that from the early uses of computers in the fifties, there have been attempts at user interface design without it being a concept yet. Such applications were mainly for air traffic control, defence and ergonomic principles [31]. Clearly, we cannot attribute the relevance of UIDs to one particular person, but we can identify the starting point when UIDs became essential and a vital component of a product's success.

The need for UID research and development became apparent in 1970 when there were already many familiar components of modern user interface design, such as pointing devices, windows, menus, icons and hypermedia [31]. Then, in 1973, the first computer to support an operating system based on a graphical user interface (GUI) appeared, the Xerox Alto. This was a groundbreaking development that gave way to the first personal computer, the IBM 5150 in 1981, then the second commercial computer, the Apple Lisa Office System 1 in 1983, and the Macintosh in 1984, which was the first commercially successful mouse-driven computer with a graphical user interface developed by Apple [36]. As a result, the user interface transitioned from a command line interface (CLI) to a graphical user interface based on windows, icons, menus and a pointing device called mouse (WIMP user interfaces), which is the legacy of Xerox PARC and was made popular by Apple [46]. In the years that followed, as the computer evolved from a specialised tool to a computer for personal daily use, the target user also changed. The non-programming user also known as the end user joined the human factor. So as commercial interest in the computer increased, the need to create easy-of-use user interfaces began. In order to create an effective user interface, developers must comprehend and analyse user tasks. As a result, models, design guidelines, and principles based on the idea of user-centred design started to emerge. At this point, the influence of technology made it possible to explore and understand user needs, but it also presented new challenges. Developers had to think about interactions techniques and all the possibilities they offer as a result of the introduction of new input devices, such as touch pens, haptic display, haptic devices, among others hardware devices. This resulted in post-WIMP user interfaces that facilitate 3D designs and improve the experience of 2D designs [46]. Then, with the emergence of the Internet and creation of the World Wide Web, web user interfaces became even more relevant today, and also drove the emergence of communities of web designers and developers who actively adopt the user-centred approach. Furthermore, standards, patterns, usability paradigms and new perspectives are always being revised to provide a better end user experience [3].

2.2 User Interface Design Tools

Today's applications are developed using some sort of user interface software tool to help design and implement the user interface. Some of the tools that were used back in the 1990s are Window managers and Toolkits, which provided a library of interactive components and a framework that made it easy to build the user interface from scratch. Then, we find event languages for direct manipulation of the interface, mapping each user action to the associated application response. Interactive Graphical tools or interface builders, such as Visual Basic, which replaced conventional code with graphical concepts, also allowed programmers to learn faster to implement interfaces with this resource. Scripting Languages, with the popularity of C and C++ new scripting languages such as Python, perl and tcl/tk emerged with the purpose of simplifying code, this allowed non-professional programmers to create sophisticated and interactive applications. Further, with the popularity of the world wide web, Hypertext such as HTML became practical. Thus, with the consistency of GUIs and the absence of changes, developers of UID tools were able to redesign new concepts and techniques that required a lower level of programming intensity to develop applications with the intention of empowering the user with easy-to-learn tools [22].

Automatic interface generation was one of the approaches of the time that did not gain wide acceptance, even though it allowed developers to design very high-level user interfaces with information about the end user. One of the reason for this, is that these tools usually used heuristic rules to choose interface details and components to build a user interface, which presented the difficulty of understanding and controlling the final result. In addition, some tools still required learning a new language in order to utilise them [22]. As the implementation and use of these tools was done by a developer, they became uncommon because a developer could create flexible user interfaces faster and with better results using conventional programming languages. However, the model-based user interface community continued to research and develop tools in this area not only for developers but also including the end users, resulting in approaches with more significant development [24]. Therefore, we can take advantage of these model-based tools that automatically generate a user interface design to comprehend and determine the requirements of an end user.

First, let us demonstrate how the three categories below might be used to group UID tools based on their level of automation in UI generation techniques they provide:

- Fully manual: In this category, a fully manual tool does not provide any mechanism to automatically generate a functional UI. In other words, the user would need significant amount of knowledge, experience in UI development as well as a strong understanding of design principles to design and implement the entire UI. Some current tools that can be considered in this category are, for example, Sketch [4], Figma [12] and Adobe XD [49] which are the three most popular tools in the design industry used for web design, wireframe, prototyping and mockups by an experienced designer [51]. There is also the Axure RP [11] tool, which can be used for the same reasons but can also be refined in terms of functionality and management requirements [50]. It is important to note that tools supporting the early stage of a user interface, like those previously stated, are frequently found in this category together with code editors and IDEs such as Visual Studio Code [20], Sublime Text [38], Netbeans [13], Intellij IDEA [16] and others.

- Semi-automatic: In this category, a semi-automatic tool can offer some mechanisms to automatically generate part of the UI, but still requires from the user to do some settings manually to complete the UI and/or give it functionality, as a result, the target user is usually a designer with at least little knowledge in programming. For example, some tools in this category are Mickey [30], Jade [52], Trident [2], Mecano [35], Mobi-D [34], XWeb [29] and DB-USE [45]. Further details about these tools can be found in Section 2.2.2.

- Fully automatic: In this category, a fully automatic tool provides mechanisms to automatically generate a functional UI, have a high level of automation to design and implement a UI with little or no manual effort on the part of the user. For example, some tools in this category are Humanoid [41], PUC [27], Uniform [23], Huddle [28], Supple [7], Arnauld [8] and Ability modeler [10]. For more information about these tools, see Section 2.2.3.

Even though it may be subjective to classify tools based on the mechanisms they provide as semi or fully automatic, we must keep in mind that the main distinction between these two categories is the effort on the part of the user to generate the UI automatically.

In this section, we focus on tools that provide semi-automatic and fully automatic generation of UIs. Although our focus is on the generation of UIs for controlling smart devices and *things*, we broaden our scope to the generation of UID in any context due to limited information on related work in the context of IoT for this particular area. Similar to the previous case, our focus is on tools for end users with no programming skills, but we also discuss some tools for experienced designers due to the information constraints.

2.2.1 Automatic User Interface Generation

Automatic user interface generation is a sub-field of human-computer interaction (HCI) that focuses on the process of developing techniques and tools for automatically generating UIs, with the goal of offering advantages like low-cost and rapid implementation [44]. Based on the research of Brad Myers [21], he introduced a categorisation of user interface tools according to how designers define the interface. This categorisation allows various automatic UI generation techniques to be classified into *language-based tools* that specify the UI syntax in a special purpose language, *interactive graphical specification tools* that allow interactive UI design, and *model-based generation tools* that generate a UI from a high-level model or specification [14, 39].

The most advanced approach for fully automatic UI generation is the *model-based* approach. This method involves using models to represent different aspects of the UI, allowing for automated generation of code, layout, and behaviour. In comparison, *language-based* and *interactive graphical specification tools* may require more manual effort to design and implement the UI. There are several models commonly used in this approach according to Schlungbaum and Elwert [39]. We will provide a quick summary of the most representative ones:

- Task model: This model represents the tasks that the end user needs to be able to perform with the system.
- Domain model: This model captures concepts, objects in a domain, usually this model is combined with others such as the task model and application model.
- User model: This model describes the characteristics and preferences of the end user. This model, like the Domain model, is also combined with others.
- Application model: This model is object-oriented and based on the services provided by the system, the purpose is to facilitate the UI behaviour.
- Dialogue and presentation model: The Dialog model describes the human-computer conversation. It specifies all the mechanism (inputs, buttons, labels, commands and others) used by the computer to query and display information to the end user. The presentation model is concerned with UIs visual representation and how objects are displayed in different dialogue states. These models are usually linked together.
- Platform model: Specifies the characteristics of the platform for the UI.

2.2.2 Semi-automatic UID tools

In this section we focus on semi-automatic UID tools for end users and developers. We provide an overview of the tools based on UI generation techniques, as well as their limitations. Because an end user can also refers to a developer, in the next sections, the term "end users" will refer to people without programming knowledge, while "designers" will describe those who possess some basic programming skills. This distinction will prevent any confusion and enable effective communication of ideas.

We start with some pioneering tools in the field of automatic generation of UI dialog boxes such as Mickey [30] and Jade [52]. Jade is a rule-based approach that makes use of predefined rules and heuristics established by user interface design principles and best practices. These rules, combined with a minimal textual specification provided by the user in an application programmer, automatically create input dialogues such as menus, buttons and dialogue boxes. The user is required to know markup languages to provide such a specification. Jade also offers a direct manipulation editor that allows to modify the output dialog with multiple look-and-feels, giving the user more control over the final graphical layout of the dialog. Jade does not add any functionality to the dialog box; instead, it focuses on creating its visual layout [52]. One limitation that is noted, but not addressed, is that Jade was created for end users; however, adjusting rules, dialog boxes and some styles that are not supported in the Jade editor requires the services of a designer who knows the syntax and format used. On the other hand, Mickey exhibits a similar behaviour where the designer's textual specification was used as the input to generate a dialog box; but this text was a new language based on Pascal, so it required the user to be familiar with a new syntax. The key of Mickey specification model is the mapping of user interface techniques to a particular Pascal constructs. The distinction between Mickey and Jade is that while Jade has multiple consistent layouts, a graphical editor, and is geared toward end users, Mickey offers a single consistent layout for all dialog boxes, lacks a graphical editor to modify the output, and is a low-level tool designed for Pascal designers [30].

Other tools that can be considered semi-automatic are tools that provide design assistance to a designer in order to generate automatic UI such as Trident [48, 2], Mecano [35] and Mobi-D [34]. These tools provide a methodology and a supporting environment with a set of techniques and tools to produce UIs. The TRIDENT Project introduces a tree-based AIO selection technique for generating a UI, this technique provides a classification of UI components based on their interactions, known as Abstract Concrete interaction Objects (AIO) and Concrete interaction Objects (CIO). These classification, are used to set rules that are then mapped by a decision tree to produce an automatic selection of the AIOs. These rules can be based on screen space, data relevance, user experience level, among other guidelines provided. For example, we can create a rule based on the user's level of experience and state that a list box would be best suited for selection by an intermediate end user, while an edit box would be best suited for an experienced end user. Some benefits using this methodology include a good visibility of rules, an easy explanation of the reasoning and backtracking; nevertheless, there are also drawbacks such as redundancy of rules and the possibility that it becomes too large [48]. In the case of Mecano, it provides a framework for assisting the development of UI interfaces aimed at designers. The strength of Mecano is the application of design rules to a domain model to generate an application but it relies on the designer to specify this domain model and review it in an editor as shown in Figure 2.1, which illustrates an example of a procedure for a medical treatment generated from a domain model, displaying a choice between x-ray therapy and chemotherapy.

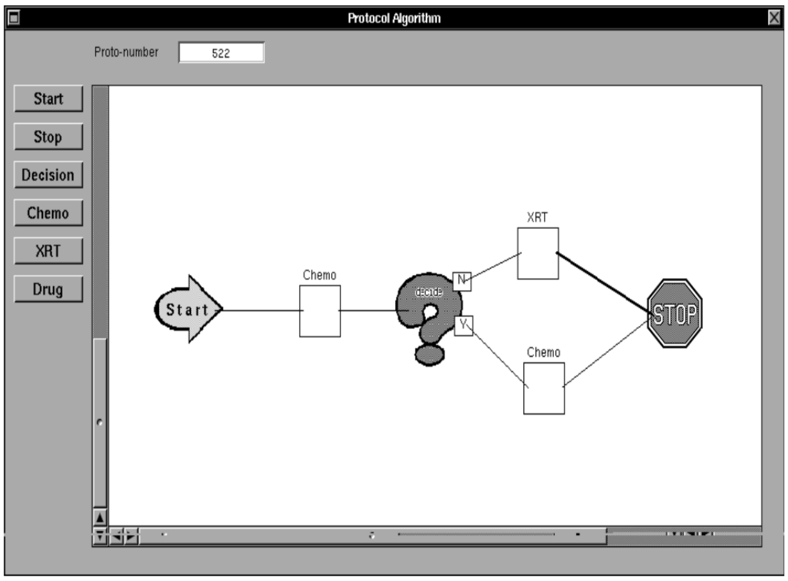


Figure 2.1: Intelligent designer tool in Mecano [27]

The domain model is used to generate windows and dialog elements as well as the dynamic behaviour of such an interface automatically. After the generation of the interface, the designer has the option of customising. This option is to involve the end user in the process. The limitations of this approach is that it is focused on domain-specific interfaces like medical forms, but it does not support generic model types automatically, so the manual effort is higher for these types [35]. Mobi-D is a successor tool to Mecano, it similarly generates form-based interfaces from domain models aimed at designers, but it differs from Mecano in that it also includes the user in an iterative loop. The cycle therefore begins with the reception of an informal textual description of the end user task. The designer will then be able to use model editor tools to define the domain model from this description. Mobi-D recommends some presentation and interaction techniques to guide the designer during design using the user-task description and the domain model. For example, if the user needs to enter a number, Mobi-D provides UI elements consistent with the model for the designer to choose from. In this final phase, the user can help the designer even more by offering feedbacks on each choice before the final UI is obtained [34]. We can immediately infer that Mobi-D solved Mecano's limitation of not supporting generic UIs, but it is still not suitable for end users.

High-level language-based approaches like XWeb [29] can also be categorised here. These approaches often involve improving existing languages to facilitate the creation of descriptions or specifications of interface requirements; however it may not always produce an optimal UI, because refining the UI is a manual process. Such is the case of XWeb, a web-based tool, that it is aimed at designers and the approach of this tool is the focus on development of a high level of semantic language to produce UIs automatically. This formal specification is based on XML. XWeb produces UIs that are adaptable to all devices and is built on a WWW architecture with his own protocol XTP to retrieve information. Their method for creating UIs starts with the definition of XViews, which are XML descriptions of elements that must be included in the UI (application model). The XViews is then mapped to possible values and interactors, which are also structured in XML trees. Interactors can be seen as a widget kit, and they must be designed manually by designers too, like the XViews description. One limitation of XWeb is that it may experience issues with screen layout on small devices and, if there are any, it must be solved manually by the designer [29].

A different approach is the DB-USE [45] tool, which combines the task, domain and user models for generating user interfaces. It is made to assist UI designers, and in addition to creating the Interface, it also creates the functionality and the actual application. DB-USE consist of five phases: model analysis, relation making, UI design, application function design, and code generation as shown in Figure 2.2, which provide an overview of the process. The process starts with three inputs: a task model in a XML file, a database that represents the domain model and a diagram file that captures the capabilities and limitations of the end user such as experience, cultural and psychological characteristics. The Model Editor will load these inputs so that the model analyst can check or modify them if necessary. The UI designer must next provide the actions task from which he wants to produce the UI in order to connect the task model with the domain model. DB-USE supports this interaction by displaying its sub-tasks in accordance with RST principles (Rules for selecting the Sub-Tasks). In the UI design phase, the UI builder automatically creates interfaces using mapping rules from the task and domain, as well as control types chosen from the CIOs (Concrete Interaction Objects) that meet the user's preferences. Then, during the application function design, the Function Editor Agent builds the functions of the application using the input from the domain model, and after that, the code generation phase produces the UI and application's code. A messages base for errors, warnings, and information to be displayed to users is also included in DB-USE.

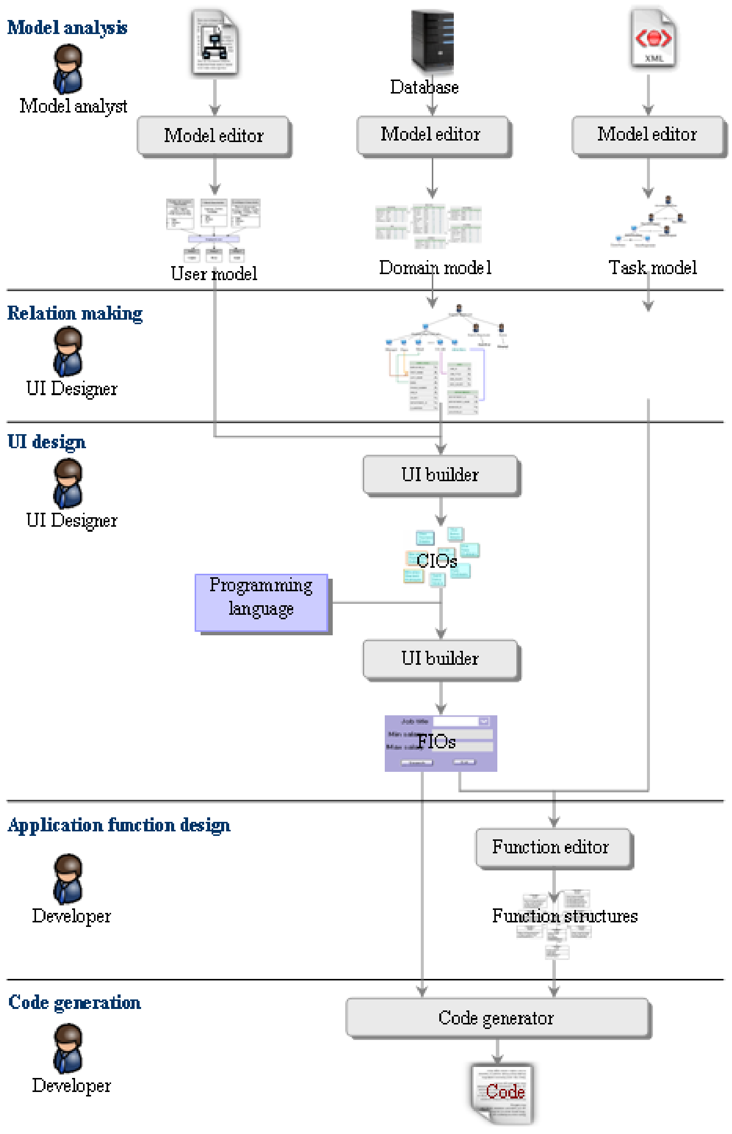
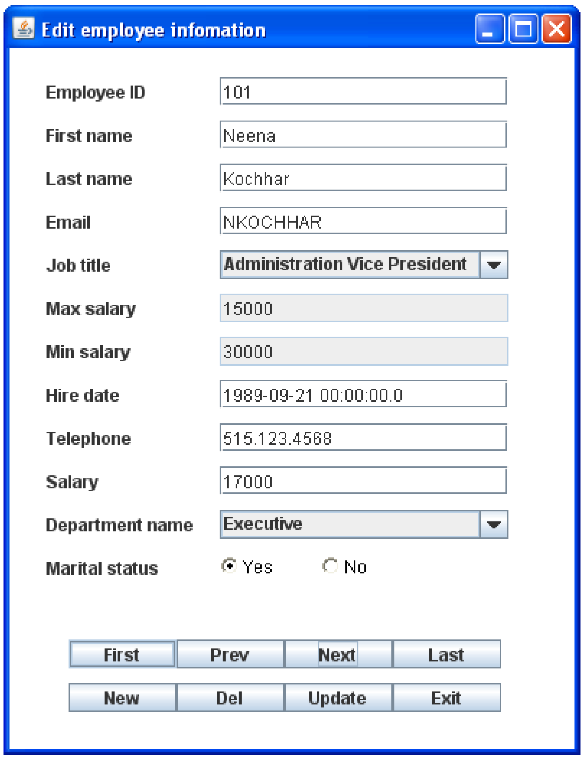


Figure 2.2: An overview of the DB-USE process [45]

Overall, DB-USE is able to produce user interfaces (UIs) and application code for carrying out database applications' generic functions, including display, add-new-value, update-new-value, delete-new-value, search-for-values, review-value, cancel-operation, and exit functions. An example of an edition generated by DB-USE is shown in Figure 2.3. The limited support for complex UIs, such as navigating between dialogue boxes, which may be necessary for some types of applications, is a limitation that is addressed for this tool. However, we can also find the dependence on designer expertise as a limitation; the designer must translate the inputs into the necessary format and depends heavily on him on configuring the mapping rules and the selection of CIOs in the UI design phase for producing good results [45].



Edit employee infomation

Employee ID	101
First name	Neena
Last name	Kochhar
Email	NKOCHHAR
Job title	Administration Vice President
Max salary	15000
Min salary	30000
Hire date	1989-09-21 00:00:00.0
Telephone	515.123.4568
Salary	17000
Department name	Executive
Marital status	[✓ Yes] No

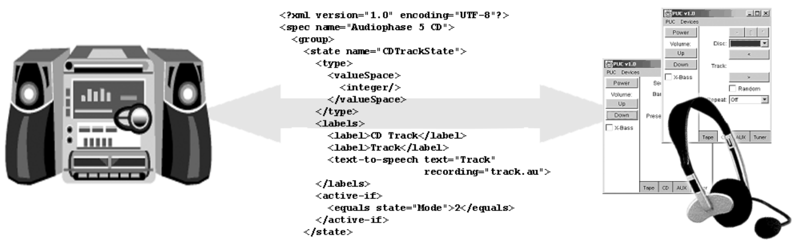
Figure 2.3: Edition of an employee information form generated by DB-USE [45]

2.2.3 Fully Automatic UID tools

In this section we will focus on fully automatic user interface generation tools, the only distinction with semi-automatic tools is that these tools can provide a complete design with all the necessary components of a user interface automatically. We will give an overview of the tools based on UI generation techniques.

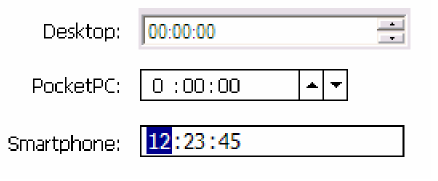
We start with HUMANOID [41], an early tool from the 1990s aimed at designers to experiment with possible scenarios and dialogues during UI development, it supports multiple levels of specification to generate a UI and it also gives functionality. The interface is specified by elaborating a semantic description with information regarding five dimensions that HUMANOID proposes: Application Design, Presentation, Manipulation, Sequencing and Action side effects. The Application Design defines the commands, objects, variables and data flow constraints of the interface. The Manipulation has a template mechanism to refine the UI step by step, in this dimension you can see the effects of the object interactions. Then, there is the manipulation, where you specify the actions and the gesture to call the action, finally there is the sequencing, where you specify the order of the different displays that are on the screen. These specifications do not have a specific order and can be called several times during the development of the UI. One limitation is that it lacks an interactive interface for creating the application description and it does not support the interface design customisation since by default it uses predetermined basic templates.

A more advanced tool is PUC [27], which is a model-based approach that generates UI for computerised appliances. PUC can automatically generate two different types of user interface: graphical and voice, based on a high-level description of a device's function as shown in Figure 2.4. This specification is xml-based and does not contain details about the appearance of the interface, which is left to the interface generator. The PUC interface generator contains extensive information about the data type of each device component, its respective tag, either in audio via a url or in text via mark-up tags, the dependency between variables and information about the similarity of user interface elements in a group tree. All of the data that the generator uses to make design decisions (using a decision tree) was gathered through a user study. A limitation about this approach is that it presents inference problems with regard to particular UI element operations that are not supported by all devices. For example, the addition of list items in an MP3 player is arbitrary while in an answering machine message the addition of a new item in the list is always at the end. PUC was not a model with a general solution [27]. A new technique was developed two years later and integrated into the PUC tool to improve its limitations, Smart template [25] technique supports a number of different representations, which allows the interface generator to create different arrangements for a layout. For example, when it recognises that an element can have several interaction representations, the interface generator together with platform-specific controls can improve the interface by choosing the appropriate control, Figure 2.5 shows different representation designs for the same Time element, and we can see the appropriate one according to the device. Smart Templates are defined in advance by template writers, who specify the different representations that a component is likely to use. This specification is presented as rendering code for interface generators and contains two parts: Control choice and Data translation. Control Choice uses rule techniques to decide which control should be rendered, while data translation translates the state values into a specific format to the chosen control, and since controls usually use the same data representation, its data translation can be shared across multiple platforms. This approach gives the impression that these smart templates are hand-coded even when they are used to automatically generate better UIs, because the tool does not offer support in developing this smart templates, it is considered to be in a conceptual level. However, as a future work the authors are planning to implement many new Smart templates and provide a list of templates for common appliances to generate UIs [25].



```
<?xml version="1.0" encoding="UTF-8"?>
<spec name="Audiophase 5 CD">
  <group>
    <state name="CDTrackState">
      <type>
        <valueSpace>
          <integer/>
        </valueSpace>
      </type>
      <labels>
        <label>CD Track</label>
        <label>Track</label>
        <text-to-speech text="Track"
                        recording="track.au">
      </labels>
      <active-if>
        <equals state="Mode">2</equals>
      </active-if>
    </state>
```

Figure 2.4: A simplified overview of PUC generating a UI [27]



Desktop: 00:00:00

PocketPC: 0 :00:00

Smartphone: 12:23:45

Figure 2.5: Representations for time using Smart Templates [25]

Another based-model approach is Uniform [26] which its main contribution is to provide consistent UI even if inconsistent specifications are provided. Uniform is developed on top of the PUC tool, so it shares the same specification language and part of the UI generation process. Figure 2.6 shows more details of this integration, where it can be seen how Uniform starts from the knowledge-base generated by PUC and also how the UI is then processed in 4 phases.

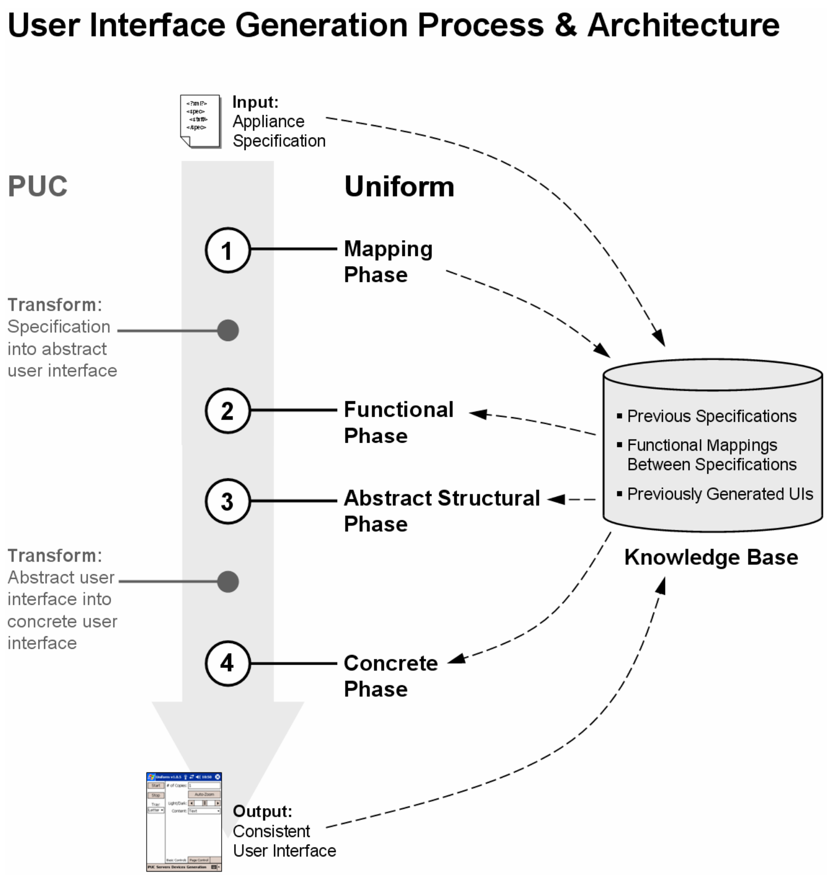

User Interface Generation Process & Architecture
Input: Appliance Specification
PUC
Uniform
1
Mapping Phase
Transform: Specification into abstract user interface
2
Functional Phase
3
Abstract Structural Phase
Transform: Abstract user interface into concrete user interface
4
Concrete Phase
Previous Specifications
Functional Mappings Between Specifications
Previously Generated UIs
Knowledge Base
Output: Consistent User Interface

Figure 2.6: Uniform generation process integrated with PUC [26]

The mapping phase compares the new specifications with those of the knowledge base to find similarities, the functional phase ensures functional consistency, the abstract structural phase ensures structural consistency and the concrete phase ensures a similar visual appearance with others. For that reason the knowledge base is important in this architecture, and the the mapping of functions from different appliances is the most relevant. Every mapping is represented by a graph and distinguishes the similarities between the appliances. For example, in Figure 2.7, we can see this mapping of functions for media control (play, stop and pause), where it shows that a Panasonic VCR has been the basis for consistency 3 times: itself, the answering machine and the DVD Player. Since Cheap VCR uses only commands, in its case it cannot be base for the others just for itself; this is represented by infinity cost. The rest of appliances count 0 because they were generated to be consistent with Panasonic VCR. Uniform includes these cost edges to guarantee consistency, in case there is no similarity with any appliance, it can traverse infinite cost edges. A limitation of this mapping is that Uniform does not know why two devices are similar or how they are related, so the mapping design rules cannot be fully automatic, it needs a better structure to receive this information [26].

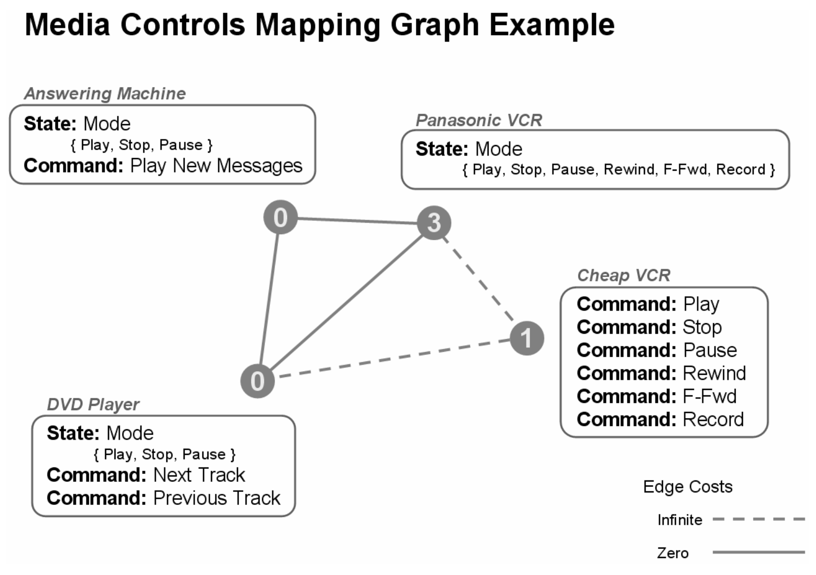


Figure 2.7: An example mapping graph of functions in Uniform [26]

Another tool that is implemented on top of PUC is Huddle [28]. This tool focuses on creating task-based UIs automatically for systems that need to connect multiple appliances. The purpose of this tool is to reduce the time and effort required to program multiple devices into a single authoring tool, like the universal remote control. Huddle requires the wiring diagram, which is currently specified manually in XML by the designer, the specifications of each device from the manufacturers, and the knowledge base of Uniform and PUC. These three inputs are necessary to generate the UIs that will enable the end user to connect their devices. Figure 2.8, provides an overview of the Huddle architecture in which we can see the three inputs and also the intermediate processes that automatically generate two types of interfaces: the flow-based interface (FBI) and Aggregate User Interfaces (AUIs). To understand the purpose of both UIs, we can look at Figure 2.9, it serves as an illustration of a home theatre setup where the FBI allows the end user to configure their appliances and set their goals on a TV and speakers, then the AUI which becomes the main interface that controls both appliances, shows the common characteristics of the content flow described in the FBI, also it does not provide the full set of each appliance functionality, but only the meaningful ones. The key idea behind this design is the use of the content flow model, a novel concept built on top of the PUC and Uniform architecture, shown in Figure 2.8.

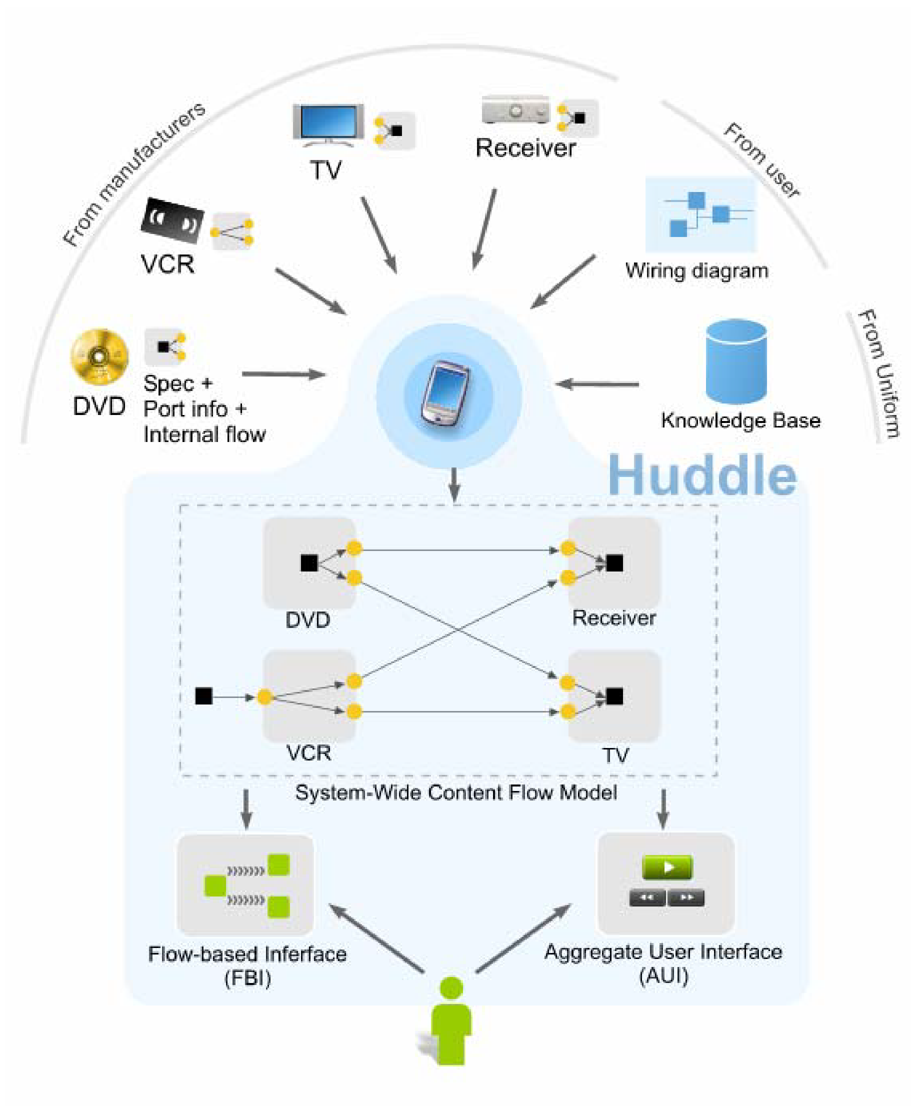


Figure 2.8: An overview of the huddle architecture [28]

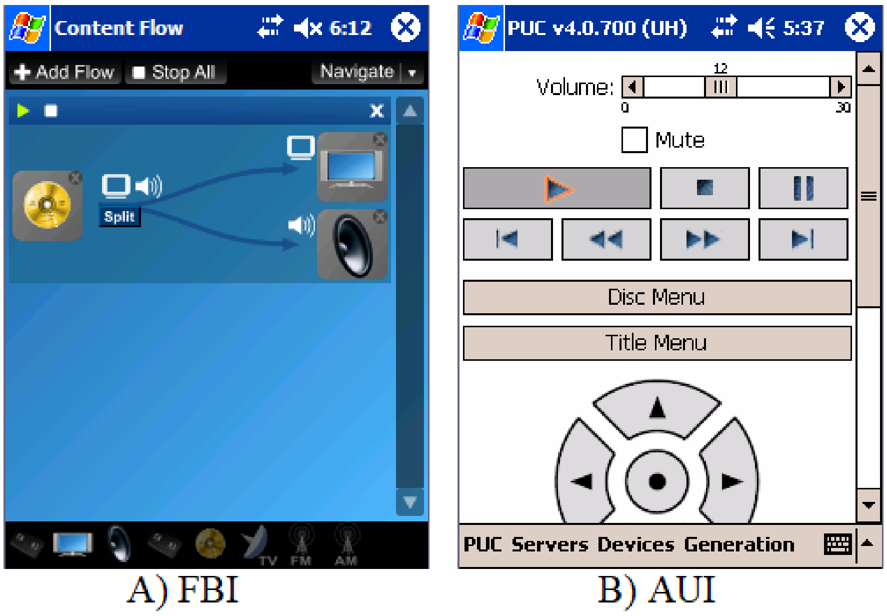


Figure 2.9: The flow based interface (FBI) and the aggregate User Interface (AUI) of Huddle [28]

The content flow contains details about each appliance, including whether it creates content (source), whether it can show material (sink), and whether it can accept and transfer content to other appliances (pass-throughs). The purpose of the content flow along with the wiring diagram, is to show how all the appliances are connected so that Huddle may figure out how to start a flow automatically. For example, the movie should only be accessible when the scan feature of the dvd is disabled. Huddle also uses the GraphPlan planning algorithm to enable the AUI. Although this tool was only proved for home theaters, some of its limitations include the limited correspondence of the content flow with some desired tasks to perform. In addition, there is no support for modeling the initial wiring description or trouble-shooting issues with it, which makes it inevitable the need for a designer with technical knowledge of appliances to configure this tool for end users [28]. Another advanced tool that goes a step further with respect to UI generation is Supple [9, 7]. The Supple tool approaches UI generation as an optimisation problem. It searches to reduce the amount of effort required by the end user to complete the desired UI tasks. Supple uses three inputs to accomplish this: the functional interface specification, which must be provided by a designer and contains the interface elements and constraints; the device model, which details the device's capabilities, including the UI widgets that are available; the constraints on those widgets; and the estimated user effort required to manipulate those widgets; and the user trace (a user model), which contains the user's manipulation patterns on a given interface element. Using these inputs, along with their rendering algorithm and cost function, Supple generates multiple UI options and selects the one with the minimum cost. Figure 2.10 provides an illustration of this process by showing a hierarchical tree that represents the functional interface specification of an application that manages a set of devices in a classroom and the UI that was generated using the algorithm, the model device, and the user traces. Overall, the key contributions are the new functional interface specification, the rendering algorithm, and the addition of user traces [7].

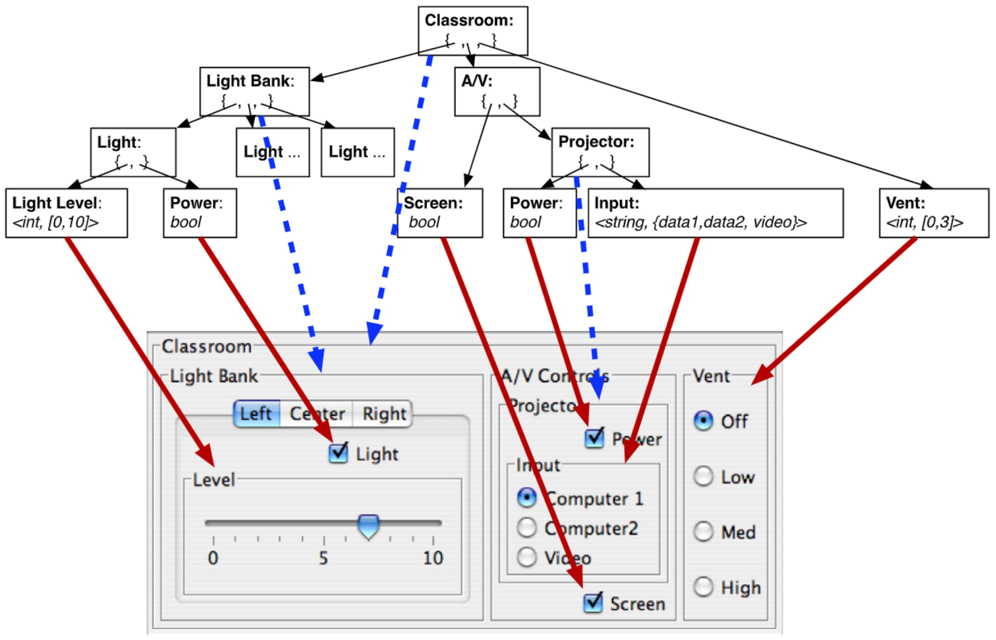
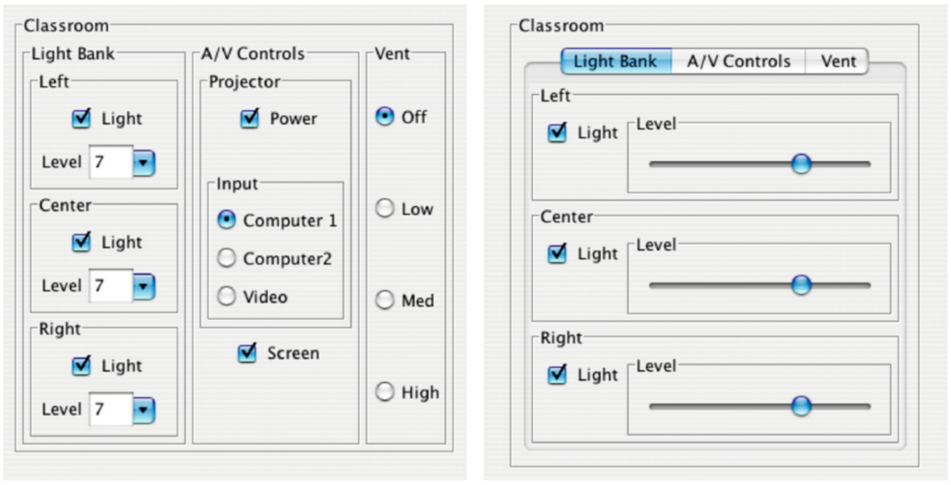


Figure 2.10: An automatically generated UI for controlling devices in a classroom by Supple [9]

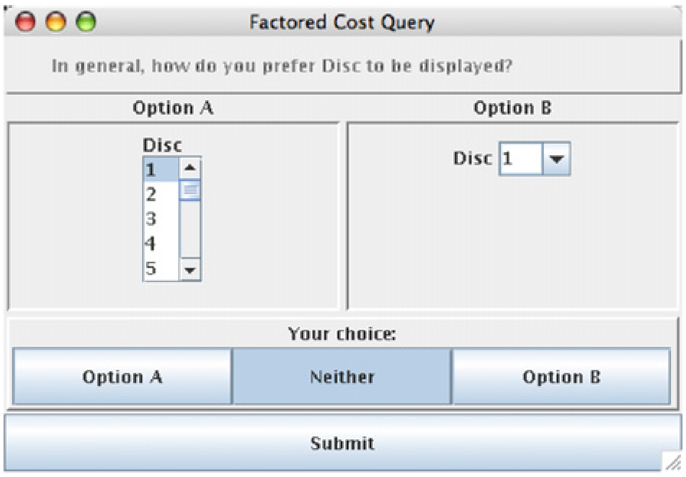
As an extension of Supple, two additional tools were developed: Arnauld [8] and the Ability Modeler [10] tools. Arnauld focuses on end users preferences and customisation, while the Ability Modeler tool is designed for end users with motor impairments. Previously, Supple's cost function had shown the possibility of capturing almost all end user preferences by giving specific elements more weight on user traces. However, using different parametrisations in the cost function generated different UIs. For instance, Figure 2.11 shows two alternative UIs with the identical inputs but using different parametrisations, the first UI favours navigation and the second UI favours convenient widgets.



A) A UI favouring Navigation B) A UI favouring convenient widgets

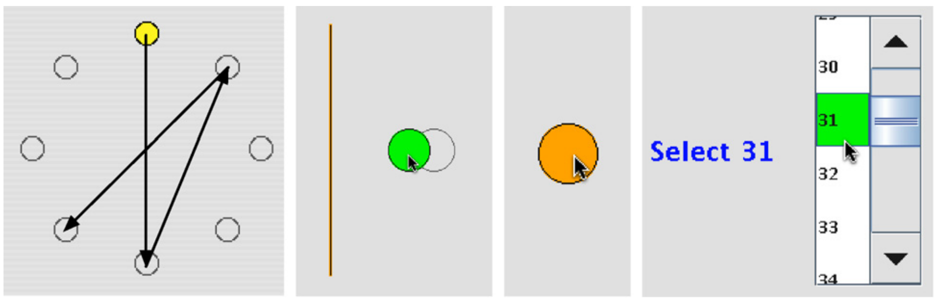
Figure 2.11: Two UI for controlling devices in a classroom with the same size constraint generated by Supple [9]

As a result, Arnauld was developed as a tool that learns the correct weights based on end user preferences expressed through concrete UI examples resulting from question generation algorithms, see Figure 2.12. A study [9] on Arnauld has shown that it can capture almost all subjective aesthetic and concerns of end users. On the other hand, Ability Modeler was developed to learn the weights of a person's motor performance by observing actions repeated multiple times such as pointing, dragging, list selection, and multiple clicking, as shown in Figure 2.13, which is the setup of how they collected data from participants with motor impairments. Figure 2.14 is an example of the comparison between two UIs generated by Supple and Ability modeler, where it can be seen how participants without motor impairments prefer lists to combo boxes and default target sizes, while participants with motor impairments prefer buttons, larger lists. Its general method can select features from a custom regression model. Additionally, Ability Modeler tool has manual options to make adjustments to the vision capability of each end user, such as increasing the size of the labels once the final UI has been generated. Some limitations of Supple and its extensions, it is the lack of support in modelling the functional interface specification, requiring the configuration of the tool by a designer before the end user can build a UI automatically. Further, despite the fact that its tree-structured interface representation makes it possible to design simple UIs like dialog boxes, it has not been tested with more complex ones like Outlook or Word [9].



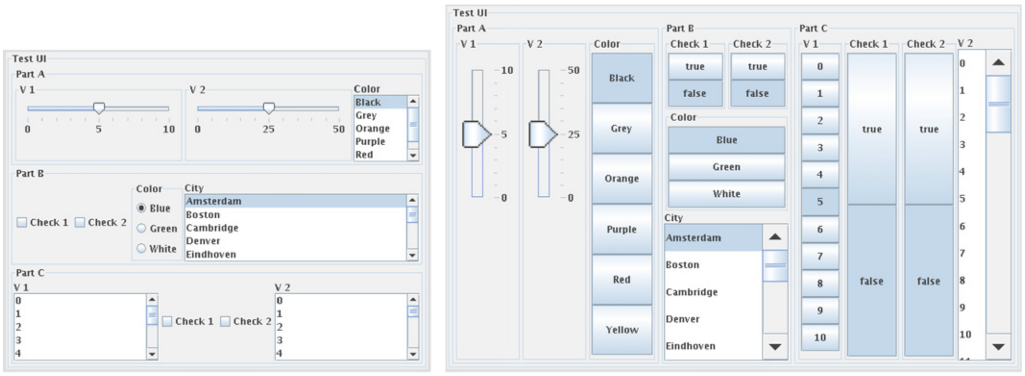
In general, how do you prefer Disc to be displayed?

Figure 2.12: An example of a question used by Arnauld [9]



A) Pointing B) Dragging C) Clicking D) List Selection

Figure 2.13: Setup of the performance of basic tasks for Ability Modeler [9]



A) Preferences based on able-bodied participants B) Preferences based on participants with motor impairments

Figure 2.14: Comparison of UIs generated by Supple based on preferences [9]

2.3 Summary and Conclusions Drawn from Related Work

We have conducted a review of various tools that offer automation techniques in the field Automatic User Interface Generation, with a focus on model-based UI tools. Between the 1990s and 2016 there was a rich body of work in this field, but as time goes on, there is a lack of new tools. We found a recent review paper identifying and analysing 28 contributions of UI design research for modelling tools between 1980 and 2017 within the GUI category, and it came to the conclusion that there are very few studies on how the design of the UI impacts the creation of conceptual models, as well as that there are few design recommendations found in most publications and that there is little focus on the usability of these tools. Future research on the impact of the design of UIDs tools is recommended by this review, along with the requirements and needs of the users of modelling tools and convincing recommendations for the design of UIs in modelling tools should be investigated [43].

From the model-based tools presented, we can observe that most of them are aimed at designers, providing them with tools and models to support the generation of UIs. Some of these tools take into account the end user in the design process (e.g.Mobi-D [34], Huddle [28], Supple [9], Arnauld [9], Ability Modeler [9]), while others rely on conventional designs for generation (e.g.PUC [27]). Both approaches have demonstrated positive results. Another finding is that we did not identify any semi-automatic or fully automatic design tools for generating UIs that provide control over IoT smart devices and things. Instead, we found fully manual tools such as , ThingWorx[1], Blynk[2], among others. It is important to mention that this is not an exhaustive review of all available tools, but rather an exploration of key concepts and techniques in automatic UI generation.

We can also infer that the process for creating an interface (UI) automatically for these model-based tools that are semi-automatic and fully automatic typically begins with a conceptual design specification of the application (e.g.Mickey [30], Jade [52], XWeb [29], Humanoid [41], Smart template [25]), the domain (e.g.Trident [48], Mecano [35], Mobi-D [34]), the functionality of the device (e.g.PUC [27], Uniform [26], Huddle [28]), or a combination of textual specifications as in the case of DB-USE and Supple that combine domain, user, and task in the case of DB-USE, or domain, user and device, such as Supple. Following this specification each tool uses different techniques depending on its requirements to design and develop UIs automatically. In Table 2.3, we present an overview of the tools with their most dominant techniques that have been identified.

Tool	**Category**	**Target Audience**	**Model**	**UI Generation Techniques**
Mickey [30]	Semi-automatic	Designer	Application-based, Rule-based	Pascal editor, Mapping code blocks, Code generation
Jade [52]	Semi-automatic	Designer	Application-based	GUI Builder, Code generation, Mapping rules
Humanoid [41]	Fully automatic	Designer	Knowledge-based, Application-based	Template mechanism, Top-down design
Trident [48, 2]	Semi-automatic	Designer	Knowledge-based, Domain-based, task-model	GUI Builder, Model editor based on rules, Mapping interaction objects
Mecano [35]	Semi-automatic	Designer	Domain-model	Model editor based on rules, GUI builder
Mobi-D [34]	Semi-automatic	Designer and End user	Task-model, Domain-based	Model editor based on task, Domain and integrating models, GUI builder, Code generator, Help generator, Design assistant, Knowledge base
XWeb [29]	Semi-automatic	Designer	Application-based, Web-based	Protocol XTP, XML tree structure
PUC [27]	Fully automatic	Designer	Device-based	Interface generator using decision trees to render, Mapping objects, Smart templates [25]
Uniform [26]	Fully automatic	Designer	Device-based	Mapping functions based on similarity
Huddle [28]	Fully automatic	Designer and End user	Device-based	Interface generator, knowledge base, System-wide content flow model
Supple [9, 7]	Fully automatic	Designer and End user	Optimisation-based, Domain-based, User-based, Device-based	Cost function, Optimisation algorithm
Arnauld [9]	Fully automatic	Designer and End user	Optimisation-based, Preference-based, Domain-based, User-based, Device-based	Machine learning algorithms, Reasoning and query generation mechanism
Ability modeler [9]	Fully automatic	Designer and End user	Optimisation-based, Domain-based, User-based, Device-based	Machine learning models
DB-USE [45]	Semi-automatic	Designer	Domain-based, User-based, Task-based	Mapping based on rules, Model editor, GUI builder, Function editor, Code generator

Table 2.1: Overview of the automatic UID generation tools

The techniques used in the UI generation process that were repeated by more than one tool are as follows:

- GUI Builder: It is a graphical editor and also called WYSIWYG editor. It allows to manipulate visual elements directly on the UI, usually by dragging and dropping elements onto a canvas. The GUI builder generates the necessary code that represents the design of the final UI.

- Model editor: it is an interactive editor that defines the data model and architecture of the system.
- Function editor: Defines the functionality and logic of the system, used to define the actions (interactions) in the UI.
- Knowledge base: It is a repository of information containing data, facts, standards and other information.
- Mapping: It is a management technique, it contains specifications that determine how data should be represented, we find mapping based on rules, code or abstract and concrete interactions objects.
- Template mechanism: It allows to reuse elements and layouts of an existing UI as a starting point.
- Code generator: It translates the design into code that can be used to build the application, the language output is directly dependent on the tool.
- Help generator: It generates documentation, tool-tips, tutorials and other useful components to guide the user in the development of the interface.

There are also techniques developed for specific tools such as the system-wide content flow model, a system using a planning algorithm developed by Huddle [28] to link the physical ports of the devices. The optimisation algorithm and cost function provided by Supple to render a UI. The protocol XTP and XML tree structure by XWeb [29]. Machine learning algorithms, Reasoning and query generation mechanism by Arnauld [8] and the design assistant by Mobi-D [34]. As they are specific techniques their limitations will depend on the tool and requirements where they are developed.

The choice of UI generation techniques can be subjective or based by related work. Often, UID tools are designed based on existing tools, by experience, or combining different model-based approaches. However, regardless of the approach taken, the user studies provide important feedback on the usability and effectiveness of the tools, which can help make the right decisions when choosing a UI generation technique. In this case Supple [9] and PUC [27], Uniform [23] provides a user study for analysis; other tools, to the best of our knowledge, do not supply this information. We can also give a general process in the context of an automatic user interface design tool based on this model-based tools. This general process is based on similar steps described in the UID tools with the exception of machine learning techniques such as Supple and its derived tools, see Table 2.3:

1. Define the UI specifications.
2. Identify and select the abstract and concrete interaction objects that will be used in the UI.
3. Map the abstract interaction objects to concrete objects.
4. Define the default layout within the UI for materialising the concrete interaction objects automatically. For instance, specifying sizes, position, alignments to avoid overlap with each other, and others.
5. Define the flow of the UI and the interactions between the end user and application.
6. Generate or Integrate the UI design with the application functionality.

Another important aspect to consider based on the related work is the functionality of UID tools. While pioneering tools like Mickey and Jade generate UI code that can be integrated into the application, modern UID tools now provide both the necessary application functionality and UI interaction capabilities to enable end users to make applications that are both robust and easy to use.

2.4 Resulting Requirements

As we discussed in Chapter 1, there is a need for end users to have more control over the design of their own user interfaces in the context of IoT authoring tools. For those without programming skills, fully automatic UI design tools offer a promising solution for creating IoT applications by empowering end users with a module to design their own interfaces, we can make it more accessible to a wider audience.

In this chapter, we analysed several model-based tools in the area of automatic user interface generation to identify the techniques and process to arrive at this automation in order to gather various functional requirements, we also relied on a research study presented by J. Wang et al [51] comparing the three most popular UID tools to derive our non-functional requirements that such module should meet. The requirements together with some research papers or studies from which they are derived are listed below. These requirements form part of the answer to our first research question (RQ1) where we define as functional requirements: R1, R3 and non-functional requirements: R2, R4, R4.1, R4.2.

Requirement 1 (R1). Support for end users to model the specification of the UI.
Motivated by the inferred general process of the model-based tools (e.g. Mickey [30], Jade [52], Humanoid [41], Trident [2, 47], Mecano [35], Mobi-D [34], XWeb [29], PUC [27], Uniform [23], Huddle [28], DB-USE [45]) for automatic user interface design tools described in the previous section. End users should be provided with a module that allows them to specify which elements, controls or components should be present in the user interface of their IoT applications that they will interact with without writing any lines of code.

Requirement 2 (R2). Generate a quickly and efficient UI design with minimal effort.
Motivated by many presented fully automatic UID tools (e.g. Humanoid [41], PUC [27], Uniform [23], Huddle [28], Supple [7, 9], Arnauld [8], Ability modeler [10]). The effort required by the end user to generate the desired user interface in accordance with their specifications is one of the differences between fully automatic and semi-automatic UID tools and we are looking to provide a fully automatic tool. Therefore, the module must be able to gather specifications with the least amount of work and rapidly generate the UI.

Requirement 3 (R3). Integrate UI design with the application functionality.
Motivated by all the fully automatic UID tools analysed (e.g. Humanoid [41], PUC [27], Uniform [23], Huddle [28], Supple [7, 9], Arnauld [8], Ability modeler [10]), where the generated user interface is aligned with the functionality and capabilities of the application. The module must be able to demonstrate the functionality of the applications created in the context of IoT authoring tools, offering a complete user experience.

Requirement 4 (R4). Provide a friendly and simple user interface easy to use.
This requirement is aimed to the user interface of the module that the end user must engage with in order to automatically generate an IoT user inter-

face (R1). The module must be straightforward, friendly, and simple to use. This requirement is subdivided into two sub-requirements.

Requirement 4.1 (R4.1). Simplicity in design.
End users who have experience with software applications in any context can find a way to interact with a complex user interface without prior training [45]. However, our target users are individuals with little to no knowledge of programming or design skills, which may group end users with little experience in IoT authoring tools. Therefore, it is important for the module to have a simple interface that allows any end user to interact with it, providing their specific needs and requirements mentioned in R1. Motivated by the UID tool DB-USE [45] that groups users based on some characteristics such as training and experience.

Requirement 4.2 (R4.2). The module should be of quality.
There are not many studies that concentrate on the usability and user experience of UID tools because the interest is in the evaluation of the generated user interface rather than the tool at that initial stage [51]. However, our interest is that the module (R1) not only generates the desired user interface but can also provide satisfaction in usability and user experience, so we will use the Post-study System Usability Questionnaire (PSSUQ), the same evaluation used by J. Wang et al. [51] to identify the strengths and weaknesses of the module with respect to System Usefulness, Information Quality and Interface Quality.

3 Background

This chapter will focus on our second research question (RQ2) along with the principles and heuristics required to satisfy the requirements of the our first research question (RQ1).

We first give a comprehensive overview of the eSPACE authoring tool, which will be the solution that we will upgrade by including our module to allow fully automatic UID. Next, we will discuss the eSPACE reference framework we must use, to create a user interface (UI) for controlling smart devices and *things* along with its respective functionality.

The chapter concludes with the section on guidelines and heuristics, which presents new concepts we must consider in our module as well as solutions and principles to adhere to in order to meet the functional requirement R1 and the non-functional requirements R2 and R4 of RQ1.

3.1 The eSPACE Framework

In the previous chapter, we presented and analysed several UID tools for generating user interfaces automatically, which we classified as either semi-automatic or fully-automatic. However, we have found that there are no fully-automatic tools available specifically for end users within the context of an IoT authoring tool. Instead of beginning from scratch to solve this problem, we start with a semi-automatic tool, which is the eSPACE authoring tool [36].

The Smart PlACE (eSPACE) authoring tool is an existing IoT authoring tool that unifies cross-device and IoT user interfaces under the eSPACE framework enabling end users to control differents smart devices and *things*. Its framework uses appropriate abstractions meant for designers or developers to simplify the UID process. Based on this, the developer can create a user-friendly authoring environment that can be used by end users to author their own UIs and applications.

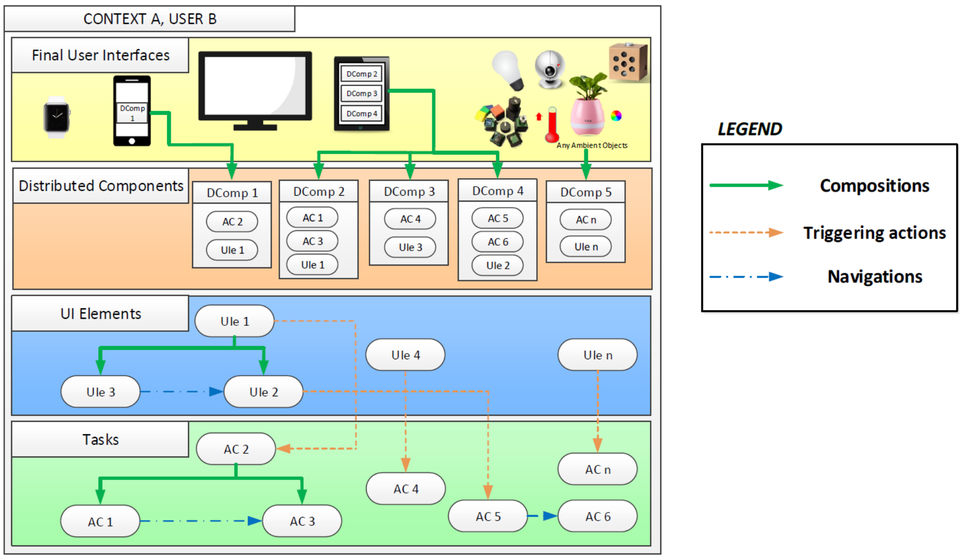


Figure 3.1: The eSPACE reference framework [36]

The eSPACE framework is composed of 4 layers as shown in Figure 3.1, it follows an approach that enables the development of new compositions and functionalities that can be extended over time:

- Task layer: It contains active components (AC), which are actions that are part of an application. Examples of these actions are adjusting the brightness of the TV, turning the light on or off, browsing the mobile gallery, among others. An important feature of this layer is that the ACs can be linked to each other to build more complex applications by reusing code, such as browsing the mobile gallery and displaying it on the TV.

- User interface elements layer: It contains UI elements (UIe) such as buttons, text fields, containers, checkbox, sliders, navigation controls, among others. Like ACs, UIes can also be linked with other UIes to form more complex UI elements. For example, to create a video player,we can combine a play, pause, and stop button UIe with a video frame UIe.

- Distributed components layer: It contains Distributed Components (DComps), consisting of at least one UI element, one AC element, or both, in which case they are linked to form rules. The rules can be considered as configurations in the application, such as turning off the room light when the user turned on the TV. As a result, an application can be made up of one or more DComps. The particularity of DComps is that any DComp can be distributed across different devices and applications.

- The final user interfaces layer: It contains the application aspect of the Final User Interface (FUI) which is a DComp assigned to a device. Alternatively, it may also be composed of multiple DComps. As illustrated in Figure3.1, it is noteworthy that an application could encompass multiple FUIs, and these FUIs are created on smart devices equipped with display capabilities.

Employing this framework, the eSPACE authoring tool presents a valuable solution for end users seeking to control all their smart devices and *things* on a single platform that can cope with their individual needs. This solution empowers end users to create and customise IoT user interfaces for their applications through a GUI builder as shown in Figure 3.2 which corresponds to the UI design view where the end user drags and drops UI elements onto the canvas. Then, to provide the desired functionality, the end user moves to the interaction view, as shown in Figure 3.3, and using the same technique, the end user drags and drops the smart devices and *things* onto the canvas to subsequently link them to add functionality to the created UI elements. Another possibility for linking functionality to these buttons is to navigate to the rules view, which offers a more textual interface. In the rules view interaction rules are created by formulating IF-THEN statements, as depicted in Figure 3.4.

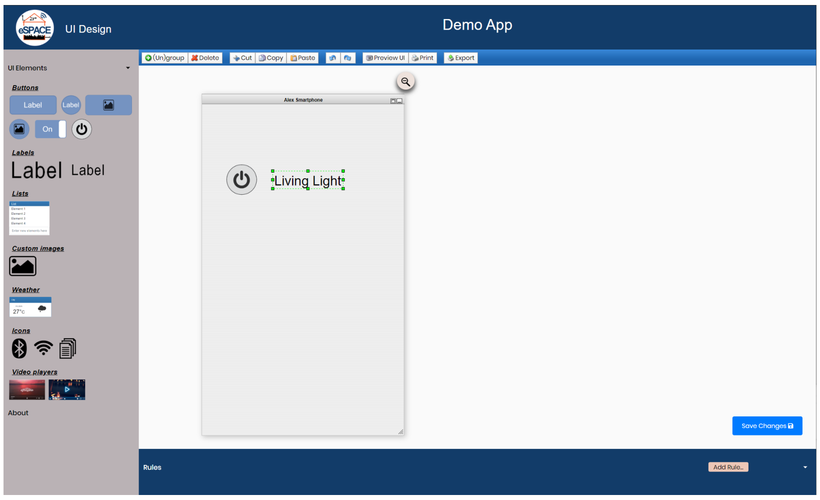


Figure 3.2: eSPACE authoring tool - UI design view [36]

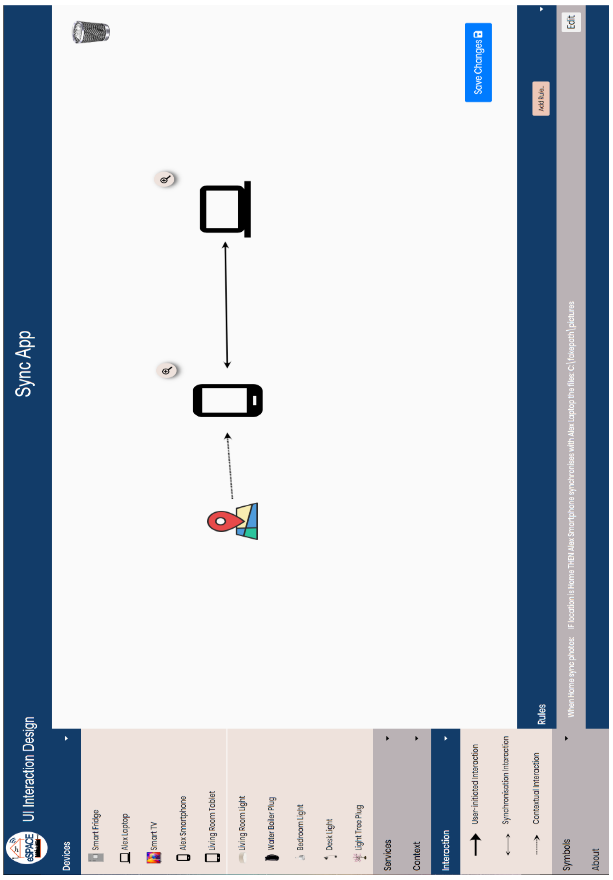
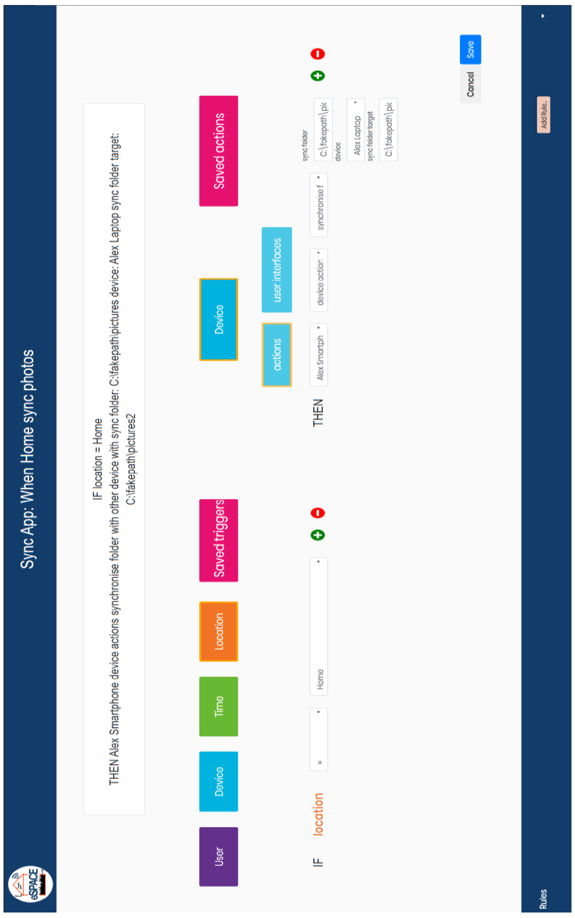


Figure 3.3: eSPACE authoring tool - interaction view [36]



IF location = Home
THEN Alex Smartphone device actions synchronise folder with other device with sync folder: C:\fakepath\pictures device: Alex Laptop sync folder target: C:\fakepath\pictures2

Figure 3.4: eSPACE authoring tool - rules view [36]

3.1.1 The eSPACE App Creation Process

In the previous section we introduced the eSPACE framework and its respective elements which we should focus on in order to upgrade the eSPACE authoring tool. In our specific case it is not necessary to modify the existing architecture and framework, but we must concentrate on how the applications are structured to recreate the development of an IoT application with its respective functionality.

In Figure 3.5, we provide an example of a welcome home application to illustrate how to apply the eSPACE reference framework layers. In this example, the welcome home app is used on the smartphone, which requires the creation of a final user interface (FUI), the devices on which we can create the FUI will be called screen devices. This application controls a smartTV (TV DComp) and a smartLight (Light DComp) by switching their states from "on" to "off" and vice versa. To control this action (AC) a button (UIe) is used which will trigger the action by a click.

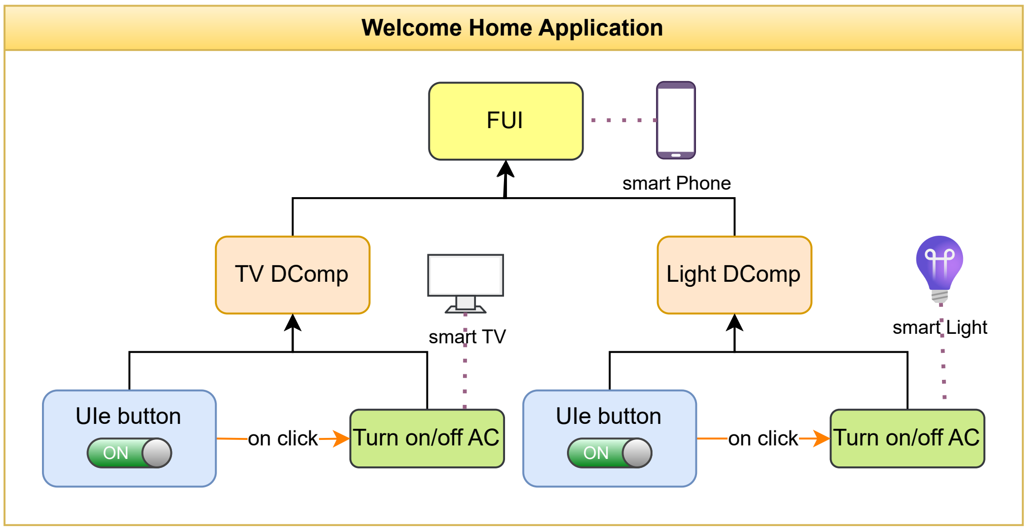


Figure 3.5: Simplified eSPACE model for a welcome home application

The process of the Welcome Home Application example within the eSPACE authoring tool begins with the creation of the FUI on the Smartphone. First, the user goes to the UI design view, illustrated in Figure 3.2. In this view, the user has full flexibility to design the user interface on top of the smartphone by creating the corresponding UI buttons (UIe). Then, after designing the UI, the user creates the TV DComp and Light DComp in the interaction view or rules view, which is shown in Figure 3.3 and Figure 3.4. When users define interaction rules in the interaction or rules view to assign a functionality to each button, eSPACE is generating a model which links the UI element (UIe) to an active component (AC) (which provides the selected functionality) and saves this composition into a DComp. Once the DComps are created they form a part of the FUI and the user can interact with this FUI to trigger the desired interaction on their smartphone.

Although the steps within the eSPACE authoring tool was not thoroughly explained, it is clear that creating an IoT application requires several steps over different views and demands some effort for the configuration of each created UI button, not to mention that it relies on basic knowledge in UI designs to create a well-designed UI. As a result this tool can be considered as a semi-automatic solution within the context of UID tools.

For our purpose and requirements, we need the eSPACE authoring tool to level up and become a fully automatic UID tool for the generation of IoT applications.
To showcase the current functionality and features of the tool, in Figure 3.6, we present the plan of a hypothetical smart home that was designed for its evaluation. The house plan includes all the smart devices and *things* configured in the eSPACE authoring tool together with their status. It is important to mention that this house plan has only been created for participants to see what happens in the smart home when using their self created applications.

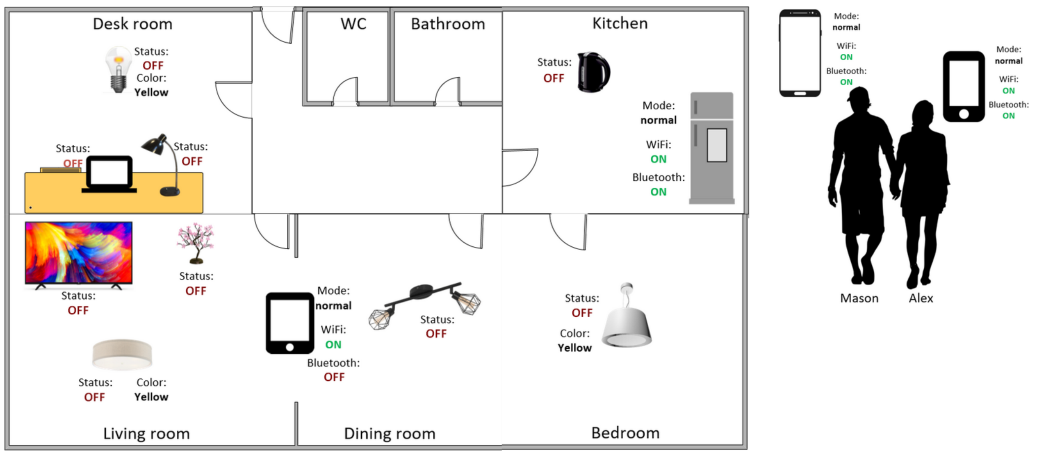


Figure 3.6: The eSPACE plan house: Simulation of a smart home

3.2 Guidelines and Heuristics

Our requirements R1 and R2 describe the need to provide a module that allows the end user to specify UI elements, controls or components with minimal effort and obtain the result quickly. Previous research in fully automatic UID tools introduced many techniques and guidelines for providing a module that takes as input this UI specifications. However, those UID tools aimed at end users require an initial configuration by a designer. Our goal is to give end users complete control without the need for a specialist.

The closest solution and the one that motivates us, is found in ARNAULD [8], where dialog boxes are used to gather user preferences as shown in the Figure 3.7.

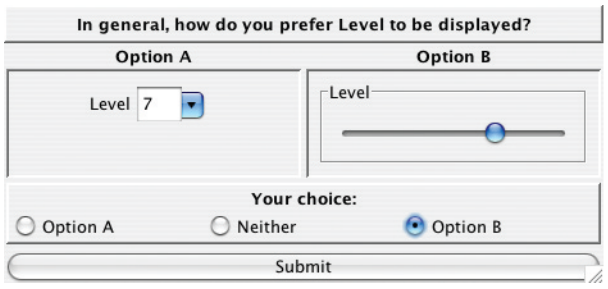


Figure 3.7: Preference for light intensity control used by Arnauld [8]

Although this approach in ARNAULD serves more as a complement to better understand the end user before beginning the UI generation process. We can extend this method to not only gather user preferences but also user requirements and to make sure that the module does not present usability problems, we can follow the basic principles of UI design [6] listed below:

- Design guideline 1 (DG1): Make common elements (e.g. buttons, text fields, checkboxes, sliders, progress bars) work in a predictable way.
- Design guideline 2 (DG2): Keep interfaces simple. Show only elements that serve a user's purpose.
- Design guideline 3 (DG3): Maintain high discoverability (e.g. shadows for buttons).
- Design guideline 4 (DG4): Respect the user's eye and attention (e.g. proper alignment, avoid including colours or buttons excessively).
- Design guideline 5 (DG5): Minimise the amount of actions (clicks) required to complete tasks.
- Design guideline 6 (DG6): Put controls near objects that users want to control.
- Design guideline 7 (DG7): Provide feedback (e.g. messages after the user performs an action).
- Design guideline 8 (DG8): Use conventional UI design patterns (e.g. breadcrumbs design pattern).

- Design guideline 9 (DG9): Maintain consistency.
- Design guideline 10 (DG10): Provide next steps which users can deduce naturally.

We already have the starting point for designing our module, but we still need to incorporate another concept used in fully automatic UID tools, which is the definition of a default layout for the UI they generate. The default UI for our particular case would be the FUI in accordance with the eSPACE reference framework and the same can be generated for any device in which an interface can be created such as smartphones, smart TVs, laptops, and other devices. Because of this, depending on these devices, the size of the UI elements, the available space and the minimum size at which end users can still view them must be taken into account.

We found some web resources [15, 5] stating that the minimum possible font size for desktop and smartphone should be 16pt. We also found a study of a responsive web design on a TV [33] indicating a minimum of 22pt for smart TVs and a study on the influence of the layout usability for a smartwatch [19] pointing 12pt as a better font size but we did not find information on other devices. This search leads us to a solution proposed by Matthew James Taylor [42], an expert website developer since 1997, who proposes the values Table 3.1, as a fluid font size according to the corresponding screen width. These values are calculated with the function calc(15px + 0.390625vw), where vw stands for the viewport of the screen width.

Screen width	Font Size
< 320px	12px
320px	16px
768px	18px
1024px	19px
1280px	20px
1536px	21px
1920px	23px
2560px	25px

Table 3.1: Minimum font sizes [42]

We adopt this solution to set the minimum font size for each device. In essence, our approach sets the minimum font size as `calc(15px + 0.00390625 * width)`. However, as explained above, the font size for smartwatches is 12px, a value we do not obtain based on the function. Consequently, we have adjusted the Table 3.1 to adapt this setting to the screen widths of less than 320px that smartwatches may have. Now that we know the minimum font size, the heuristic for determining the maximum font size we will use is to first identify the minimum font size and from there add 1pt as long as there is space available until the maximum is reached.

Finally, we can build some different default layouts for the FUI by using these values and adhering to the principles of UI design described above. Chapter 4 will go into further detail on the default layout.

4 Design and Implementation

The following chapter presents the proof-of-concept prototype for the generation of IoT applications as a result of the derived requirements from the research of the related work and the eSPACE reference framework following the principles and heuristics defined in the previous Chapter 3. Our prototype has been constructed to answer our third research question (RQ3), which will be the focus of a user study and whose evaluation is covered in depth in the Chapter 5.

We start by explaining the modifications that must be made to the current eSPACE authoring tool, then move on to the front-end of our prototype, its implementation and a use case that demonstrates its functionality. Finally we end with a summary and discussion section that outlines some considerations and limitations about the tool.

4.1 Generation of IoT Applications

We will add our module to the eSPACE authoring tool to create our proof-of-concept prototype of the generation of IoT Applications, which means that we will adopt its reference framework and existing functionality introduced in the Section 3.1. For the present module we do not need to modify the back-end of the tool. Instead, we will concentrate on a new fully automatic module that extends its current features by including a new option to generate applications following the principles described in Section 3.2. As a result, we have to make some modifications to the current application creation process and adjust it to replace the UI design and interaction processes that are involved and turn it into a single fluid process.

4.1.1 App Generation Process

We illustrate the flowchart of the app generation process in Figure 4.1. This "Generation of the app" starts by requesting the name and description of the app, then requests the screen devices that will be used in the app, for which the final user interface (FUI) will be created. The eSPACE framework can contain several screen devices for the same application, therefore our first loop begins:

1. For each screen device selected, it requests for the smart devices or *things* that the user wants to control with that particular screen device. At this point there is no distinction between screen devices and *things*, we therefore refer to it as device. Then starts our second loop:

   1.1 For each device or *thing* selected, it requests the actions (ACs) that the user wants to perform, followed by a question where users can choose to customise the representation (UIe) of the action or skip this step. In case that the customisation is chosen, then the third loop begins:

      1.1.1 For each action, it requests to select the preferred representation (UIe).

   1.2 For each screen device selected, the user is requested to select their preferred FUI between two possible choices.

After this final loop is finished, the process can be redirected to three different views. One option that returns to the main view, another option that opens the application generated and the last option in case some manual adjustments are needed on a particular FUI, it redirects to the UI design view.

After defining this process, we moved on to designing the user interface of the module in accordance with the guidelines and heuristics laid out in the Section 3.2.

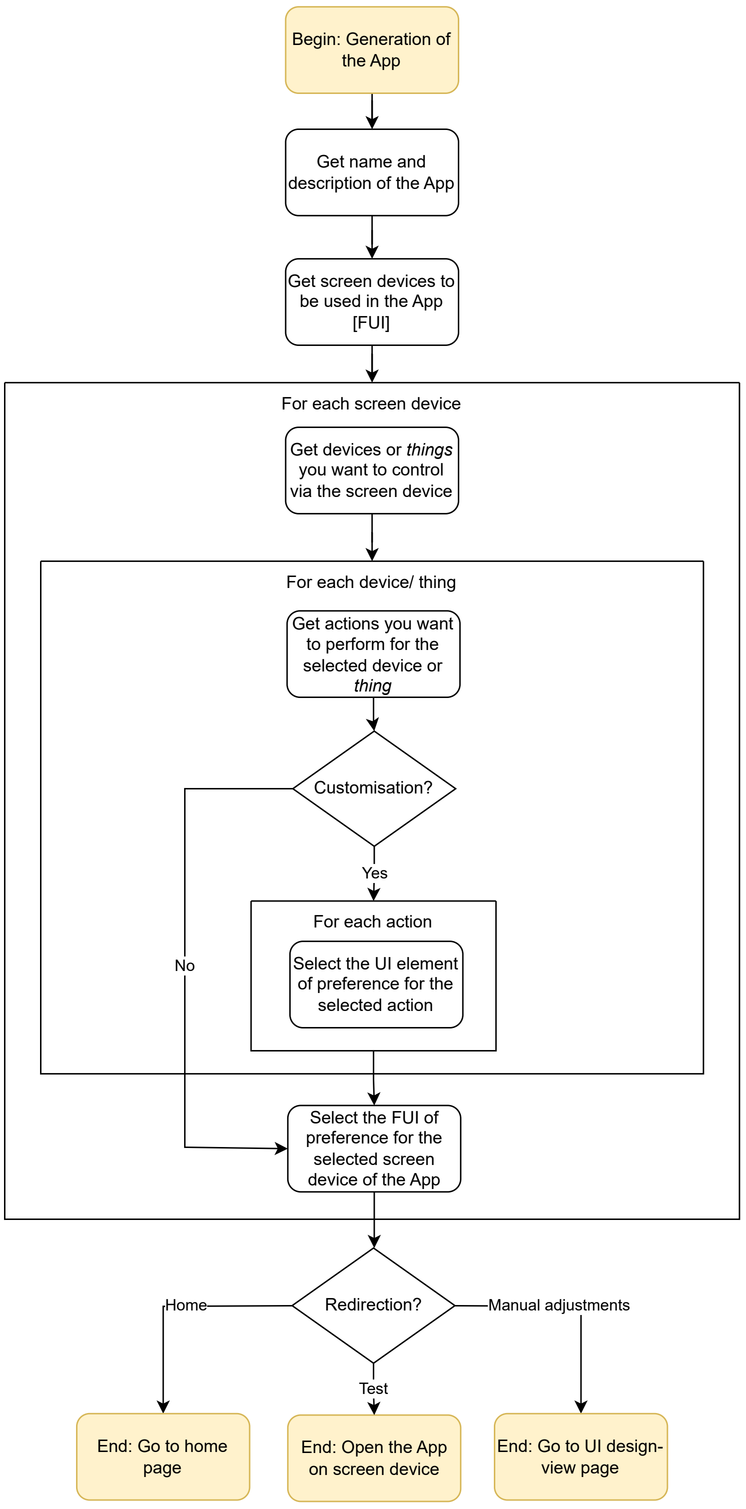

Begin: Generation of the App
Get name and description of the App
Get screen devices to be used in the App [FUI]
For each screen device
Get devices or *things* you want to control via the screen device
For each device/ thing
Get actions you want to perform for the selected device or *thing*
Customisation?
Yes
No
For each action
Select the UI element of preference for the selected action
Select the FUI of preference for the selected screen device of the App
Redirection?
Home
Manual adjustments
Test
End: Go to home page
End: Open the App on screen device
End: Go to UI design-view page

Figure 4.1: Flowchart of the generation of IoT Applications

4.1.2 Front-end Design

We briefly discussed ARNAULD's method in Section 3.2 to gather user preferences and how this solution serves our purposes. Therefore, we propose to expand this solution to gather not only user preferences but also user requirements to generate an IoT application, as shown in Figure 4.1. Our module based on this dialog box solution is presented in Appendix A, which contains the mockups that we produced according to the app generation process as a multi-step form wizard, following the basic UI design principles listed in the Section 3.2.

4.1.3 Default IoT Application Design

There are several forms to represent the look and feel of the IoT Application and the more elements and/or components it includes, the more options there are. In this initial stage of our module, only two templates are being provided as options for the user to choose their preferred final user interface (FUI). The elements that are considered in the final user interface are the following:

- Name of the App: A text represented with a label.
- Name of the device or *thing*: A text represented with a label, with the purpose of grouping the actions that can be performed for the smart device or *thing*.
- Action: The text of the action represented with a label and the value of the action represented as a button, slider, checkbox or other UI element chosen by the user.

With these elements, we create the following two templates shown in Figure 4.2, as the default options for the FUI. The first template, option A, seeks to centre all elements vertically; while the second template, option B, divides the actions into two columns, with the last action being centred if the number of actions is odd. Both templates maintain the same element size, following guidelines provided in Table 3.1 and the heuristics described in Section 3.2.

During the creation of these templates, we followed the basic principles outlined in Section 3.2. However, DG5, DG8, and DG10 were not considered, as the FUI is intended to be a single screen that controls all the desired smart devices rather than a multi-stage process.

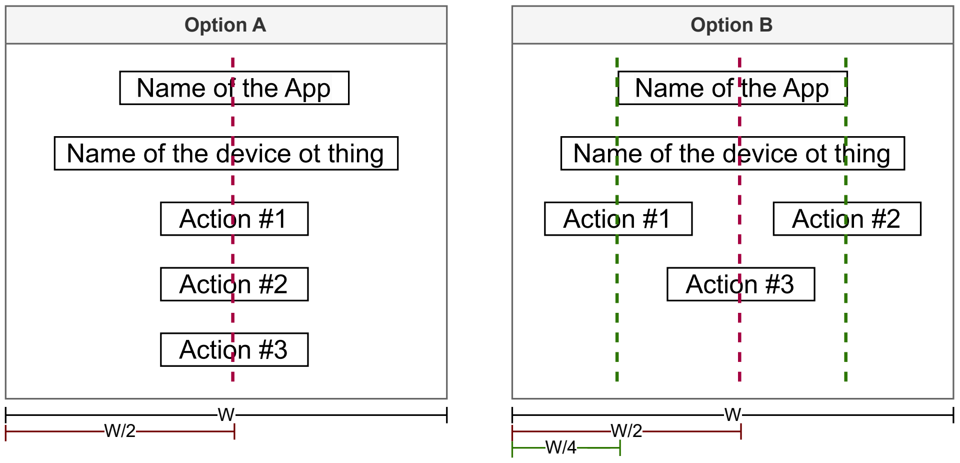


Figure 4.2: Default templates for the Final User Interface

4.2 Implementation

4.2.1 Technology

The eSPACE authoring tool prototype is implemented as a web application using the Spring Boot framework. As back-end it uses Java, and as front-end it uses JavaScript, HTML5, CSS3 and additional libraries for app creation such as canvasutilities and mxGraph. Our module uses the same front-end technologies to recreate a multi-step form wizard shown in Appendix A.

4.2.2 Generate App View

We added the "Generate App..." option, which leads to the extension of our module, to the main page of the eSPACE authoring tool. The module view includes its own HTML page with all the tags that define the page and the feedback messages, shown in Appendix A, with a separate stylesheet (CSS file). The primary colours of the tool are preserved so that it does not resemble an external option, as illustrated in the Figure 4.3, which is the starting point of the Generate App view. We used JavaScript to dynamically create blocks of HTML code and incorporate them into the HTML page, to represent the elements involving loops within the flowchart from Figure 4.1.

As some steps of this view do not require the use of additional libraries and the code is relatively simple, in this section, we focus on the more complex implementations details we had to deal with.

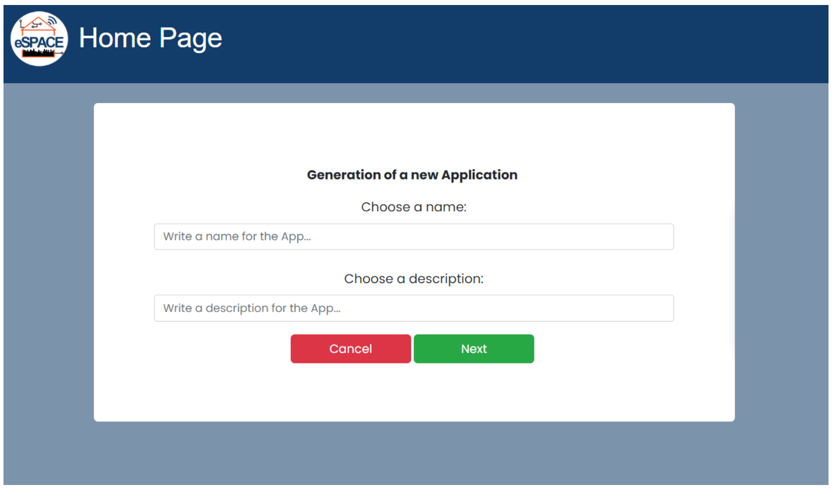

eSPACE
Home Page
Generation of a new Application
Choose a name:
Write a name for the App...
Choose a description:
Write a description for the App...
Cancel
Next

Figure 4.3: Generate App view: Start

One of the complexities that we had to deal with was the representation of the actions. Although we only used the UI elements that the tool had, the problem was that conventionally not all representations were suitable for a certain action. For example, to turn on or off a light, the checkbox is not a conventional representation for that effect. Consequently, a designer must decide which elements are better than others. During the analysis of the related work, a technique known as Smart templates [25] was introduced. It was developed on top of PUC [27], dealt with the same situation, and used this technique to help its interface generator chose the appropriate control. Based on that study we can use the same approach, employing Smart Templates using XML.

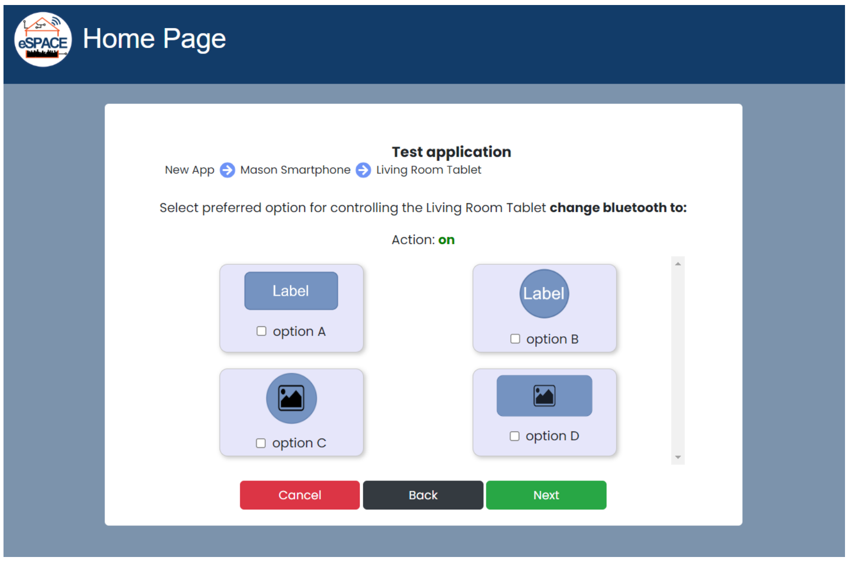
We then define our own XML language to group the UI elements, in order to identify the suitable representation to execute an action. For example, when executing actions that require adjusting a value such as changing the intensity of a light, adjusting the speed of a ventilator, it is preferable to use representations such as sliders or counters. This grouping is specified by the tag <group> that we can see in lines 3, 13, and 19 of the Listing 4.1, the attribute `group-type` specifies the category that we established, in this case `default`, `toggle-buttons`, and `adjust-buttons` respectively. These categories were created based on our observations of the tool's most common actions. The `adjust-buttons` groups UI elements that are suitable to adjust values, `toggle-buttons` groups UI elements that are suitable to switch states back and forth, and `default` category was created in case a particular action could not be captured by the previous ones, such as change the mode to silence, not disturb or loud.

Finally, the UI element is defined with the `<UIe>` tag, as seen on line 4, and the attributes required for this definition are the `type`, which determines what type of UI element it is. The `label`, which is the name used in the module to display the UI elements. The `icon`, which is the image that represents the UI element. The `id`, which is an identifier. The `require`, which can be text or image depending on the HTML attributes of the UI element, for example a button needs a text or image to give it context. The `style`, which is the CSS used for the UI element appearance in the module. The `customStyles` and `customLabel`, based on the base code of the tool, which are the categories used to identify the UI element. To retrieve the data from this configuration file, we used AJAX, and to provide an illustration, Figure 4.4 represents the `default` category.

```xml
1  <?xml version="1.0" encoding="utf-8"?>
2  <groupTemplate>
3    <group group-type="default">
4      <UIe type="button" label="option␣A" icon="/images/UIe/labelButton.png" id="UIe1" require="text" style="width:120px;padding-bottom:5px;padding-top:7px;" customStyles="labelButton" customLabel="Label␣Button">
5      </UIe>
6      <UIe type="button" label="option␣B" icon="/images/UIe/labelRoundButton.png" id="UIe2" require="text" style="width:73px;padding:4px;" customStyles="roundLabelButton" customLabel="Label␣Round␣Button">
7      </UIe>
8      <UIe type="button" label="option␣C" icon="/images/UIe/imgRoundButton.png" id="UIe4" require="image" style="width:73px;padding:4px;" customStyles="roundimageButton" customLabel="Image␣Round␣Button">
9      </UIe>
10     <UIe type="button" label="option␣D" icon="/images/UIe/imgButton.png" id="UIe3" require="image" style="width:120px;padding-bottom:5px;padding-top:7px;" customStyles="imageButton" customLabel="Image␣Button">
11     </UIe>
12   </group>
13   <group group-type="toggle-buttons">
14     <UIe type="button" label="option␣A" icon="/images/UIe/imgRoundButton.png" id="UIe4" require="image" style="width:73px;padding:4px;" customStyles="roundimageButton" customLabel="Image␣Round␣Button">
15     </UIe>
16     <UIe type="button" label="option␣B" icon="/images/UIe/powerRoundButton.png" require="text" id="UIe17" style="width:73px;padding:4px;" customStyles="powerimageButton" customLabel="Power␣Round␣Button">
17     </UIe>
18   </group>
19   <group group-type="adjust-buttons">
20     <UIe type="slider" label="option␣A" icon="/images/UIe/slider.png" id="UIe19" require="text" style="width:210px;padding-bottom:6px;" customStyles="slider" customLabel="Slider␣Element">
21     </UIe>
22   </group>
23 </groupTemplate>
```

Listing 4.1: Snippet of choice-buttons.xml example for grouping UI elements

With this method, we can continue to create new groups, change some representations, or even modify the option's name without having to make changes directly in the code view.



Test application

New App → Mason Smartphone → Living Room Tablet

Select preferred option for controlling the Living Room Tablet **change bluetooth to:**

Action: **on**

Figure 4.4: Generate App view: Representation (UIe) for the action

The implementation of the FUI was another issue at the time, and its solution will be explained in two parts. The first part, which is the implementation of the FUI, will be shown later in Section 4.2.3. In this section, we assume that the FUI is already generated and we continue with the discussion of displaying the two default FUI.

Once the process to gather all the user's requirements and preferences is completed, the two options for the FUI are generated and hidden in HTML code blocks in the page. At this point, to display both options we use the HTML2Canvas JavaScript library to take a screenshot of the HTML content of both FUIs so that we can move it and resize it as needed in the current step, as demonstrated in Figure 4.5. We provide a fragment of this implementation in Listing 4.2, where it captures the contents of option A, the first default FUI, and resize it to a scale Factor of 300, which is the size of the container from Figure 4.5.

```
var canvasOptions = {
    // Set the width and height equal to the original content height
    width: pageContainerChoiceA.offsetWidth,
    height: pageContainerChoiceA.offsetHeight
};

html2canvas(pageContainerChoiceA, canvasOptions).then(function(canvas) {
    var imageContainer = document.getElementById('image-container-
        choiceA-' + IdDevice);
    var resizedCanvas = document.createElement('canvas');
    var resizedContext = resizedCanvas.getContext('2d');

    // Clear the existing content in the imageContainer
    imageContainer.innerHTML = '';

    // Resize the canvas to the desired width while maintaining aspect
        ratio
    var scaleFactor = 300 / canvas.width;
    resizedCanvas.width = 300;
    resizedCanvas.height = canvas.height * scaleFactor;

    // Scale and draw the original canvas onto the resized canvas
    resizedContext.drawImage(canvas, 0, 0, resizedCanvas.width,
        resizedCanvas.height);

    imageContainer.appendChild(resizedCanvas);
    pageContainerChoiceA.style.display = 'none';
});
```

Listing 4.2: JavaScript code snippet for displaying the FUI options

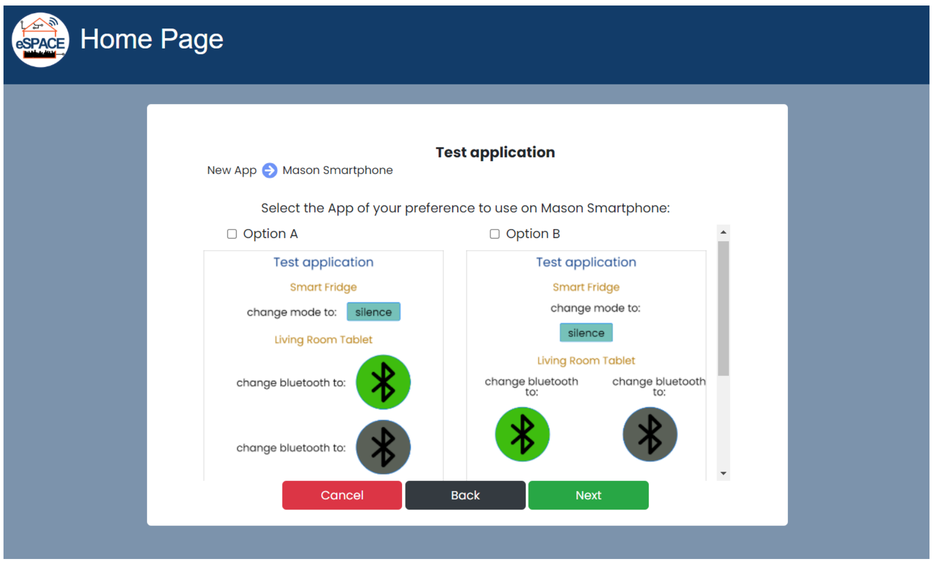
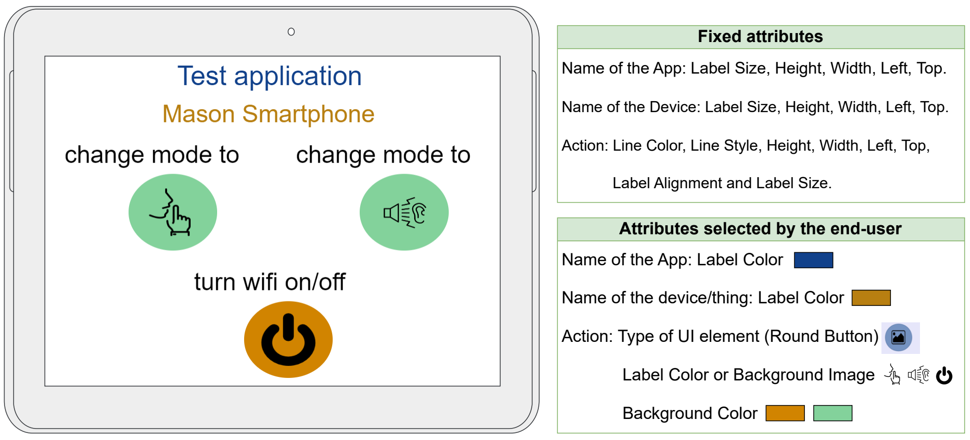


Figure 4.5: Generate App view: Selection for the FUI

4.2.3 The Final User Interface



Fixed attributes

Name of the App: Label Size, Height, Width, Left, Top.

Name of the Device: Label Size, Height, Width, Left, Top.

Action: Line Color, Line Style, Height, Width, Left, Top, Label Alignment and Label Size.

Attributes selected by the end-user

Name of the App: Label Color 🟦

Name of the device/thing: Label Color 🟧

Action: Type of UI element (Round Button)

Label Color or Background Image

Background Color 🟧 🟩

Figure 4.6: Default Final User Interface (Option B) viewed on a Tablet

In this section we focus on the implementation of the two default FUI options from Figure 4.2 that our module generates. As the eSPACE authoring tool is a web application, we used Bootstrap library to create both FUI options in HTML code. Bootstrap's predefined classes aligned with the structure of our two default templates. In Figure 4.6, we illustrate the default option B generated on a tablet, highlighting the attributes provided to the user for customisation, along with the fixed attributes whose values are assigned by the system.

Within the fixed attributes the system assigns the following default values to `lineColor` as `#2e6da4`, `lineStyle` as `solid` and `labelAlignment` as `center`. The remaining ones are calculated based on the `LabelSize` and the dimensions of the screen device. The dimensions of each screen device are retrieved from the database and to get the font size of the UI elements in the screen device, we create a function `getSizeUIelement()` shown in Listing 4.3, where it receives as parameter the screen device. It starts by getting the minimum font size in line 6 according to the Table 3.1. The `margin` variable from line 7, represents the underlying white space between elements, initially set to 10 to get a separation of 20 pixels between two elements, this value was chosen from our observations, we prioritised the focus and attention of the end user based on our content, as recommended by Mads Soegaard in *The Power of White Space in Design* [40]. After this initialisation, we call the function `getTotalPixelsChoiceA()` to get the total pixels that all elements of Option A occupy across the length and width of the screen device. This function helps us to check if there is a blank space to further increase the font size, hence in lines 13 and 14, we calculate the remaining space. If there is space in both height and width then we increase the font size by 1 and recalculate the total pixel occupancy as shown in line 22. With these values, we can calculate the new font size based on the remaining space divided by the height and width difference respectively as shown in lines 25 to 32. Finally, in line 34, we can see the case where there is no space to grow, therefore we reduce the space between the elements to a smaller margin and we keep the minimum font size of the start. Although this last case indicates that the elements do not fit on the screen, the FUI does not suffer from this because it automatically includes a scroll to display all the items. The same procedure is used for option B of the default FUI to compare it to option A's maximum font size and determine which font size fits in both templates.

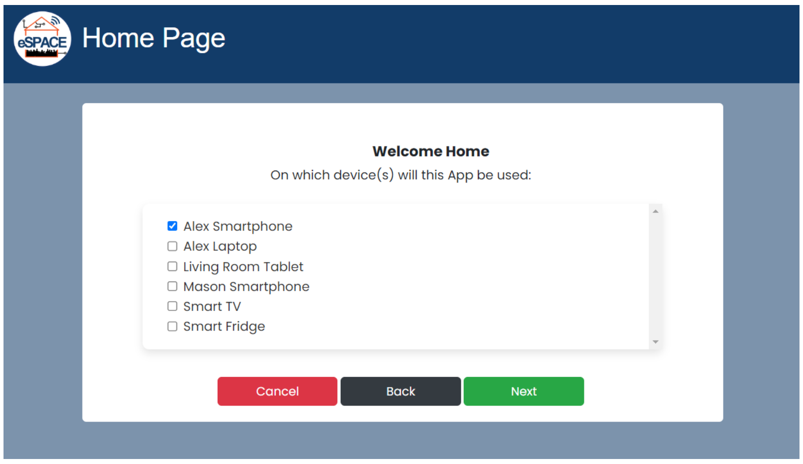
```
 1 function getSizeUIelement(screenDevice){
 2     const idScreenDevice = screenDevice[0];
 3     const width = screenDevice[2];
 4     const height = screenDevice[3];
 5     // get the minimum font size
 6     let fontSize = getFontSize(width);
 7     let margin = 10;
 8     // First: Check if everything fits in the screen
 9     let output = getTotalPixelsChoiceA(idScreenDevice, fontSize, true,
           margin);
10     let totalPixelsHeightChoiceA = output[0];
11     let totalPixelsWidthChoiceA = output[1];
12     let remainSpaceH = width - totalPixelsHeightChoiceA;
13     let remainSpaceW = height - totalPixelsWidthChoiceA;
14     if (remainSpaceH > 0 && remainSpaceW > 0) {
15         console.log("not␣overflow");
16         let temp = fontSize + 1;
17         // Second: Check if we can increase the size
18         let outputA = getTotalPixelsChoiceA(idScreenDevice, temp, true,
               margin);
19         // We get the space that sums per one size
20         let heightPerSizeA = outputA[0] - totalPixelsHeightChoiceA;
21         let widthPerSizeA = outputA[1] - totalPixelsWidthChoiceA;
22         let remainSpaceHA = (height - totalPixelsHeightChoiceA);
23         let remainSpaceWA = (width - totalPixelsWidthChoiceA);
24         let fontSizeHeightA = Math.floor(remainSpaceHA / heightPerSizeA);
25         let fontSizeWidthA = Math.floor(remainSpaceWA / widthPerSizeA);
26         fontSize += Math.min(fontSizeHeightA, fontSizeWidthA);
27     } else {
28         console.log("overflow");
29         // This does not affect the fui as a default scroll is created
30         return [fontSize, margin];
31     }
32     return [fontSize, margin];
33 }
```

Listing 4.3: JavaScript code snippet for getting the size of the UI elements

Finally, the eSPACE authoring tool had all the necessary functions accessible to manage and retrieve the data from the database through the RESTful API, which made it easier to integrate this functionality on the FUI selected by the user. It is also important to mention that because our flowchart is sequential and these functions were created for an asynchronous process using promises, we had to use the keyword `async` along with the keyword `await` within that function to pause the execution until it is resolved. As a result, the performance may suffer if we create an application that uses a lot of different smart devices or *things*.

4.3 Use Case Demonstration

In this section, we focus on demonstrating how to generate IoT Applications using our module. In a previous section in Chapter 3, we showed a Welcome Home Application example and how the eSPACE framework would be used to model it in Figure 3.5. We use this example to demonstrate its creation using our tool that is configured with the smart environment shown in Figure 3.6.



Welcome Home

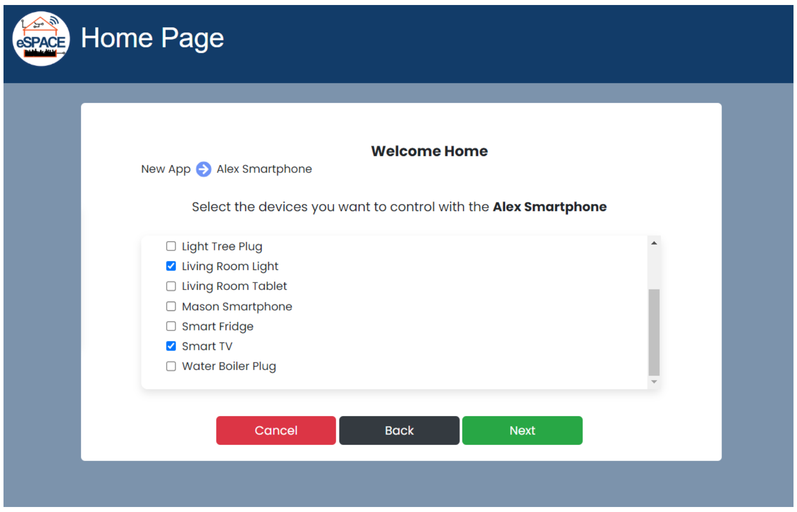
On which device(s) will this App be used:

- [x] Alex Smartphone
- [ ] Alex Laptop
- [ ] Living Room Tablet
- [ ] Mason Smartphone
- [ ] Smart TV
- [ ] Smart Fridge

Figure 4.7: Use Case: Selection of screen devices

We start by entering our module with the option "Generate App..." from the main page of the eSPACE authoring tool, then we write the name of the application and a description, in this case: *Welcome Home*. The view of this first input is shown in Figure 4.3. The next step, as shown in Figure 4.7, presents the screen devices contained in the tool and asks the user to select the device on which they will be running the application. In our example, we selected Alex smartphone.

The module then displays the message "Let's start with Alex Smartphone" as the start of the loop, and asks for the colours that we want to use in this application, by default we left the configured ones: blue for the title and mustard for other labels. Next, we must select the smart devices or *things* that we want to control with the application. Following the example, we selected the Smart TV and Living Room Light as shown in Figure 4.8. Note that we are in the initial step (1) of the first loop of Section 4.1.1.


Home Page

Welcome Home

New App → Alex Smartphone

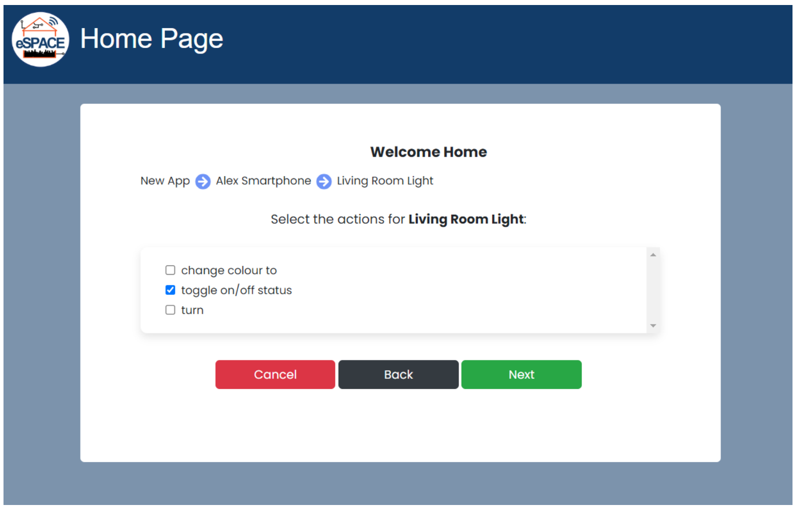
Select the devices you want to control with the **Alex Smartphone**

- [ ] Light Tree Plug
- [x] Living Room Light
- [ ] Living Room Tablet
- [ ] Mason Smartphone
- [ ] Smart Fridge
- [x] Smart TV
- [ ] Water Boiler Plug

Figure 4.8: Use Case: Selection of smart devices and *things*

The next step is to choose the actions (ACs) we wish to execute for each selected smart device or *thing*. Note that we continue with the first step within the loop (1.1) of Section 4.1.1. We choose the action *toggle on/off status* for the Living Room Light, as shown in Figure 4.9, which will give us control to turn the light on and off.

In Figures 4.8, and 4.9, we can also see a hierarchy-based breadcrumb at the top of the container after the name of the App. This breadcrumb is more informative than functional since we only display the top-level of the hierarchy which are the screen devices followed by the smart devices or *things* selected. This simplification was required because if we displayed all of the levels, the breadcrumb would not fit on a single line and would even show the complexity of the loops. After selecting the actions (ACs) for the Living Room Light, we are given the choice to either choose which buttons

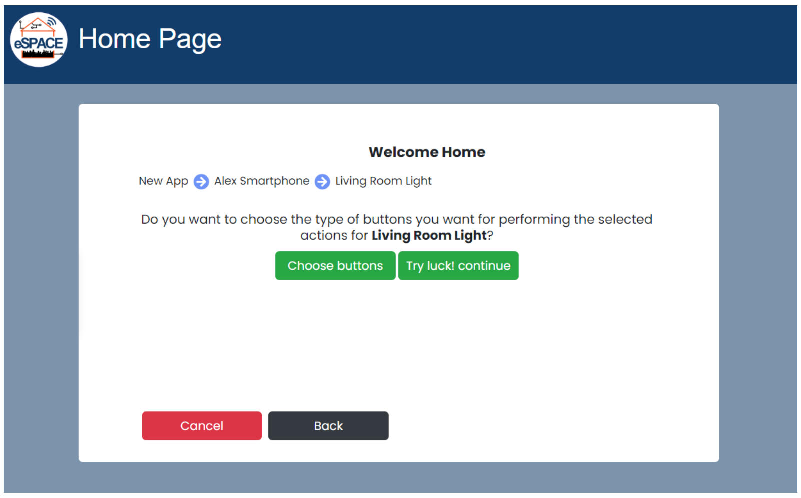


Select the actions for **Living Room Light**:

- [ ] change colour to
- [x] toggle on/off status
- [ ] turn

Figure 4.9: Use Case: Selection of actions for the Living Room Light

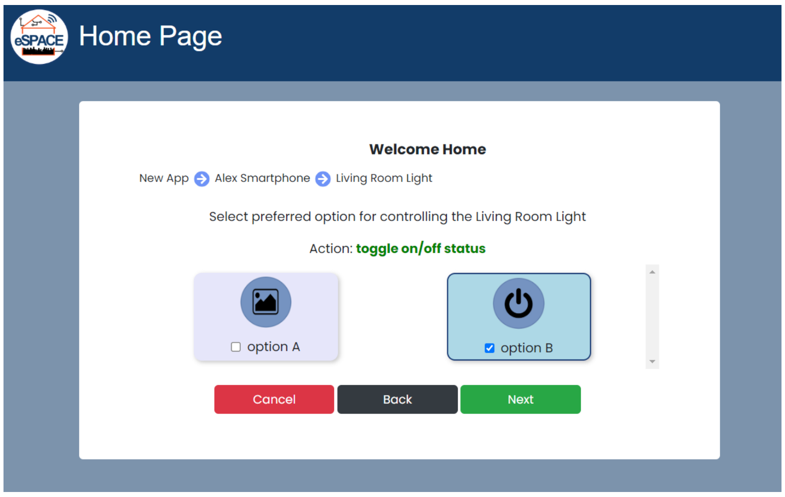
we want for triggering these actions or let the application module select these buttons for us (by pressing the `Try Luck` option). In Figure 4.10, the two options are shown. Note that this second option has been created with the intention of giving a faster alternative in the selection of the UI elements, where the system should be able to select an appropriate UI element at random depending on the selected actions. However, for this initial prototype of our module, it was left with default values. Therefore, if we select `Try luck`, a button with the text `Toggle on/off status` will be the default UI element.



Do you want to choose the type of buttons you want for performing the selected actions for **Living Room Light**?

Figure 4.10: Use Case: Customisation question for the actions selected

In this use case, we select the first option and choose to customise our application by selecting our preferred button per action. After selecting the `Choose buttons` option, we navigate to the screen shown in Figure 4.11. This screen asks us to choose between two button options for the `toggle on/off status` action. As shown in Figure 4.11, we select the power button. Once the option is chosen, then we must complete the UI element information, such as choosing the background colours of the button and the label and filling in a text to give context to the button. For this example, we left the default values of the custom style and complete with the text: on/off. Note that we are within the third loop (1.1.1) of Section 4.1.1.


Home Page

Welcome Home

New App → Alex Smartphone → Living Room Light

Select preferred option for controlling the Living Room Light

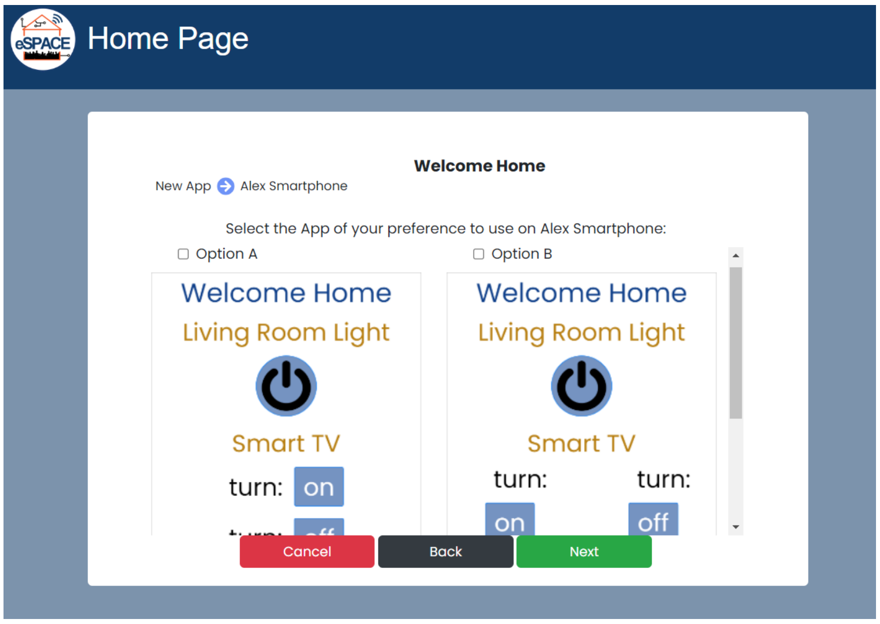
Action: toggle on/off status

Figure 4.11: Use Case: Selection of the preferred option for the actions selected

After this selection, the Smart TV goes through the same process, starting with choosing the actions to execute and the customisation for them. We selected `Turn on` and `Turn off` to control the Smart TV and `Try luck` for the customisation of the buttons. Following these steps, the module generates the application based on this set of requirements and provides us with two possibilities for the FUI of Alex Smartphone, as shown in Figure 4.12. Note that we are in the last step within the second loop (1.2) of Section 4.1.1.

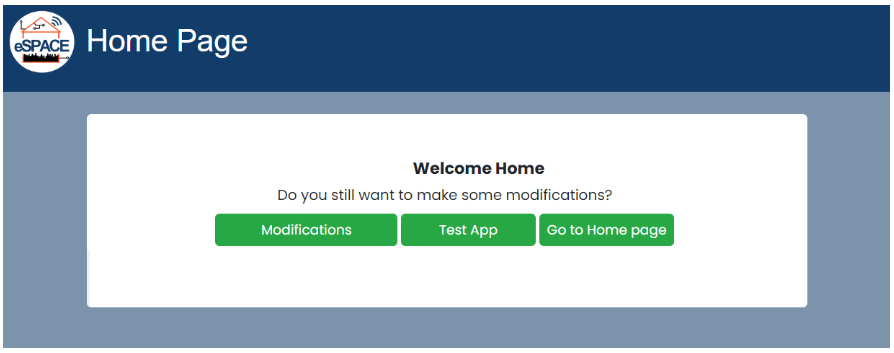
Finally, after selecting our preferred FUI, we reach to the last question of the module, which redirects to a different view from which we may either continue editing the FUI (`Modifications` option), go to the home view (`Go to home page` option) or open the application (`Test App` option), as shown in Figure 4.13.

The `Modifications` option opens a dialogue box asking to choose the FUI to be modified, as several FUIs can be created in the same application. Once the FUI has been selected, we are then redirected to the UI design view from Figure 3.2. We present the Figure 4.14 to show the generated IoT Application for the Welcome Home where we chose `option A` as the preferred FUI.



Select the App of your preference to use on Alex Smartphone:

Figure 4.12: Use Case: Selection of the preferred option for the FUI



Do you still want to make some modifications?

Figure 4.13: Use Case: Question to redirect to another view

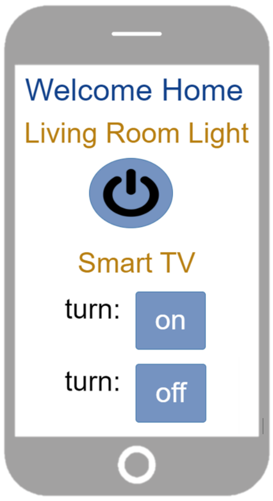


Figure 4.14: Use Case: Welcome Home Application viewed on a Smartphone

4.4 Summary and Discussion

We implemented the proof-of-concept prototype of the generation of IoT Applications on top of the eSPACE authoring tool, following the basic principles of user interface design presented in the Section 3.2. We discuss each Design Guideline (DG) in order to demonstrate how it has been validated.

DG1: Make common elements work in a predictable way. The only interactive elements used throughout the design are buttons, scrollbars and multiple select options. We respected their behaviour to not confuse users.

DG2: Keep interfaces simple. All displayed elements give a purpose to the user, such as the name of the application, the options to choose and the buttons to continue, cancel or go back steps.

DG3: Maintain high discoverability. We used conventional buttons, we kept the scroll-bar visible, and we incorporate hover effects before and after a user action for high discoverability.

DG4: Respect the user's eye and attention. During the implementation of the mockups, we received feedback from the supervisor of this thesis, which helped us to refine some of the designs. These were directly adapted to the prototype. The first refinement is from the third image of the Figure A.3. It shows all buttons aligned at the same height beneath a question for the user, because of this, the two options for that question lose focus. this is fixed directly in the prototype in figure 2. Another change was made to the buttons used to select the preferred option for a certain action, as seen in the first and second image in Figure A.4. we changed it to checkboxes to be consistent with prior selection methods, this change is shown in Figure 2.

DG5: Minimise the amount of actions (clicks) required to complete tasks. To minimise the number of clicks, the options within the multiple select are visible by disabling the show/hide property with the exception of the action selection for each smart device or thing as it contains a multiple select inside another multiple select which can take up a lot of space, in this case only the options of the first multiple select are visible.

DG6: Put controls near objects that users want to control. All designs are compact without pretending to overload with options, the focus is on providing a straightforward experience without unnecessary complexities.

DG7: Provide feedback. We present a message at the start of each loop depending on the options selected to inform the user of the step they are in. We also incorporate error messages in case the user forgets to fill or select an option required.

DG8: Use conventional UI design patterns. We use a breadcrumb navigation design pattern ilustrated in Figure 4.4 and Figure 4.5 to show the current location; however this navigation lacks redirection capabilities because the exhibited link trail is summarised due to the complexity of the hierarchy.

DG9: Maintain consistency. To harmonise with the tool, we use the primary colours of the eSPACE authoring tool.

DG10: Provide next steps which users can deduce naturally. We use the conventional colours and templates for the buttons such as green for "Next", red for "Cancel" and grey for "Back" buttons. They are centred in the middle after each user action to be performed, as shown in the Figure 4.5.

In addition to the design guidelines, we also discuss how our tool fulfill the resulting requirements (R) derived from related work in Section 2.4.

R1: Support for end users to model the specification of the UI. Specifying the elements of the IoT user interface is possible with the "generate App" option integrated in the eSPACE authoring tool, where end users simply have to click on the elements they want to control. The Section 4.1.1 details the specifications required for building the IoT user interface. With our prototype, end users can develop their IoT Applications without any programming or design knowledge.

R2: Generate a quickly and efficient UI design with minimal effort. Our prototype is designed as a step-by-step form to break down the complex process shown in Figure 4.1. With this series of manageable steps we intend to prevent end users from getting lost in the loops. This method, which was inspired by ARNAULD [8], a fully automatic tool, typically requires less effort for end users compared to a long form, but we will get more insight about this statement in Chapter 5.

R3: Integrate UI design with the application functionality. When end users interact with the Final user interface (FUI), the functionality is presented to them. The eSPACE authoring tool provides a simulation of a smart home plan introduced in Section 3.1.1, that responds to the end user's actions on the IoT Application. This requirement is verified by the end users during the evaluation of the prototype.

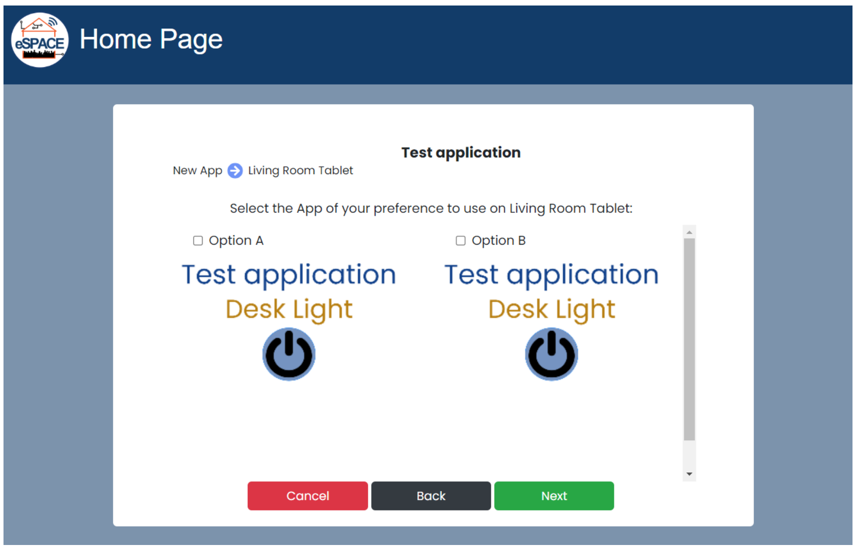
R4.1: Simplicity in design. This requirement is completed following the design guidelines described above on the usability of the user interface of our prototype.

R4.2: The module should be of quality. At this stage of implementation we cannot claim that our prototype is of high quality even if it follows the design guidelines on usability, the end users should be the ones to judge the quality. Therefore, this point is covered in detail later in the Chapter 5.

We end this section by stating the limitations of our prototype that we had during the implementation. One of the limitations we encounter is with respect to the eSPACE authoring tool. Since it is still a prototype, the tool is not completely finished as not all the UI elements (UIe) that it contains in the UI design view are functional. In this case, the tool offers the option to incorporate sliders, checkboxes, and buttons (with text or images) when choosing UI elements. However, sliders and checkboxes cannot yet be linked to functionality preventing their use in our module.

We now switch to our current prototype as we move on to another limitation. There is a lack of differences between the two default FUI templates (Option A, Option B) that it generates in a specific case. This problem arises when only one action to execute is chosen. Both templates share a common heuristic, that when there is only a single UI element then it is placed in the middle. In a future version of our prototype it should be evaluated if it is possible to design another template for a single selected action or whether the absence of a distinct design in these circumstances simply results in the generation of the only version that can be produced without a question in between. This new design not only has to be different from the first option, but it must also be appealing to the eye. We provide an example in Figure 4.15 that illustrates this scenario and one approach is to move the device name such that it is at the same height as the button as shown in Figure 4.16. However, it remains to be seen whether this new combination harmonises with the addition of more smart devices.

Finally, we plan to design a scenario where users do not face these limitations that may affect the results, but rather focus on the usability of our prototype and get better understanding of unexpected results.



Select the App of your preference to use on Living Room Tablet:

Figure 4.15: Default Final User Interface templates for an action

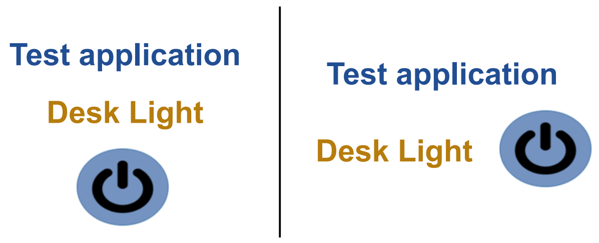


Figure 4.16: Different text alignment for the FUI of a single action

5 Evaluation

In this chapter we present the evaluation of our proof-of-concept generation of IoT applications integrated into the eSPACE authoring tool. First we present our participants who took part to the user study, followed by the protocol explaining our user study presentation, tasks execution and a post-study questionnaires prepared for the evaluation.

Following the user study we analyse the results obtained from the responses of the participants in the post-study questionnaires and our findings from process, observations and interviews conducted during the evaluation. Finally we end this chapter with a summary of our findings and some valuable insights to consider for future work.

5.1 Setup

The user study was conducted in two different locations: first, in our apartment building in which we gave participants direct access to our computer so they could use the system, and second, via a remote connection to our computer because of the unavailability of some participants during the day. We configured the fictitious smart home setup on our computer, shown in Chapter 3, for the participants to use their IoT applications made using the eSPACE authoring tool. Participants are expected to perform the user study without prior training on the eSPACE authoring tool and without a tutorial available. More detailed information is provided in the following subsections.

5.2 Participants

There were 10 participants in total, the majority of them were chosen from our apartment building, people we are familiar with. The remaining participants were university colleagues from outside our study programme, including 3 men, aged 27 to 58 years (M=35.1, SD=9.3). 7 participants indicated that they have used Samsung SmartThings and Amazon Alexa as tools for controlling their smart devices showing a certain level of expertise with smart home authoring tools. However, none of the participants had any prior experience in UI design or programming, and none of them had developed their own IoT applications before. Therefore, all of the participants were representative of our target audience. Also, it is necessary to mention that some participants were given access to the evaluation via remote connection because most of them work during the day and were only available in the evenings.

5.3 Protocol

The eSPACE authoring tool was configured and installed on our computer, with our home environment running as localhost. All tasks assigned to participants were completed on our computer, and after completing a series of tasks, they were asked to complete a questionnaire and provide final feedback.

5.3.1 User Study Presentation

We started with a short oral introduction about IoT applications and a description of the scenario shown in Appendix B.1, we used the same scenario as the author [36] because we wanted to make use of the initial configurations such as the smart devices that were already entered into the system. The explanation was available throughout the course of the study.

5.3.2 Tasks Execution

Our system was evaluated by 10 participants, they were asked to complete two tasks as shown in Appendix B.2, which involved using the eSPACE authoring tool to automatically create an IoT application. The first task aimed to familiarise them with the tool without any prior instruction or training, and the second task aimed to use the options that were not selected in the first task, so that they would have a full understanding of what the system offers. Finally after completing the tasks they were asked to complete a post-survey shown in Appendix B.3, and a short interview, which will be described in more detail in the following subsection.

5.3.3 Questionnaire and Interview

Participants were asked to fill out a questionnaire comprising of three parts as follow:

1. The Personal Information questions that help us know a little bit more about our participants such as age, gender, highest degree and whether they have experience with home application authoring tools as shown in Appendix B.3.1.

2. The Post-Study System Usability Questionnaire (PSSUQ) v3[1] as shown in Appendix B.3.2, which consists of 16 7-point Likert Scale questions to measure the system Usefulness (SYSUSE), Information Quality (INFOQUAL) and the Interface Quality (INTERQUAL).

3. The Post-Study Informative Questionnaire aims to evaluate if the resulting requirements introduced in Section 2.4 were fulfilled; however, since the PSSUQ already includes items that measure this requirements, we have focus on gaining more detailed insights regarding potential improvements in information and interface quality. The resulting questions can be found in Appendix B.3.3.

These questionnaires were transferred using Google forms[2] in order to gather the responses from the participants. Finally, the participants were briefly interviewed to know their last thoughts and/or recommendations on the system and each study task they completed.

5.4 Results

In this section we first analyse the results of the PSSUQ then the informative questionnaire and we will finish with some observations made during the user study.

[1] https://uiuxtrend.com/pssuq-post-study-system-usability-questionnaire/
[2] `https://www.google.com/forms/about/`

5.4.1 Questionnaires

The results of the PSSUQ measure the satisfaction perceived by our participants when using our system and as mentioned before it consists of 16 questions; however for our specific case the question 7 and 9 from Appendix B.3.2 do not apply in our user study. Participants cannot rate question 9 because we do not provide any documentation or online help for using the system. Regarding question 7, the system displays error messages when a participant forgets to choose an action or fill out a text box. This was the case for 3 participants, while the rest of the participants did not encounter this messages. Consequently, as they had no basis for rating, we asked them to give a neutral score. For this reason, both questions are not considered in these results to prevent affecting them.

The resulting Table 5.1, shows the overall score of all the answers of our participants as well as the 3 sub-scales or categories for system usefulness, information quality, and interface quality. We also provide the means determined by Sauro and Lewis [37] involving 21 studies and 210 participants for a better interpretation.

PSSUQ	Sauro and Lewis [37]	Avg Score
SYSUSE	2.80	1.58
INFOQUAL	3.02	1.55
INTERQUAL	2.49	2.07
Overall score	2.82	1.67

Table 5.1: PSSUQ results

To interpret the results of PSSUQ, we have to remember that the lower the score the better the performance and higher the perceived satisfaction, being 1 the minimum score that can be reached. Observing the Table 5.1, we have an overall score of 1.67, an indicator that the end user is satisfied in general with the system. The average score for the system usefulness is 1.58, the score for information quality is 1.55 and for interface quality we scored an average of 2.07. Indicators that allow us to infer that our system has a good performance. If we compare with the means of Sauro and Lewis [37], we observe even better results. However, it is important to know that the results are based on a small size of 10 participants who we are familiar with, so this combination could introduce biases. A more detailed visualisation is presented in Figure 5.1, where we can see the error bars for each category and question representing the standard deviation (S.D) of the data over this group of participants. We can observe that the error bar for Interface Quality (INTERQUAL) is the one with the highest variability considering only the final average scores; yet, because this is a prototype in its early stages, the results are encouraging to continue with further improvements.

It is also interesting to analyse the results per question to see in detail which aspects should be improved. These results are shown in Figure 5.1, where the top rated questions are Q10, Q11 and Q12 related to information quality which tells us that one of the strengths of the system is the clarity, completeness, and accuracy of how the information is presented, while the worst rated questions are Q8 related to information quality and Q13 and Q14 related to interface quality.

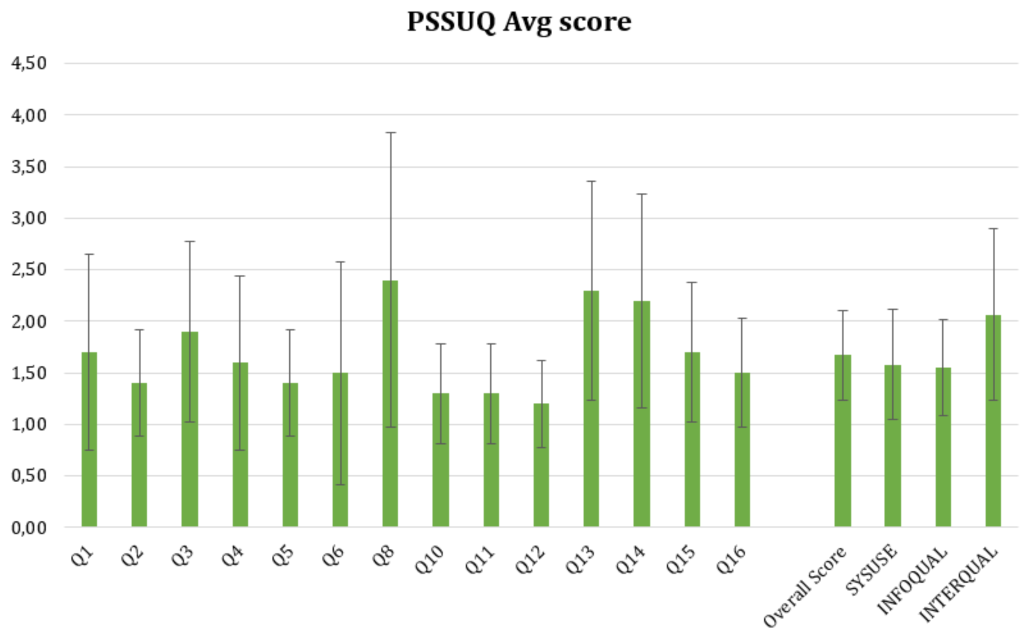


Figure 5.1: Avg score results per question and PSSUQ categories

The Q8 refers to the ease of recovering from mistakes made while using the system. It received an average score of 2.40, which generally suggests good performance. However, it was observed that some participants found it annoying when they forgot to select an action or smart device and reached the final stage. They expressed frustration about having to go back and redo the steps. Some participants mentioned that if they could create their own applications instead of following the Task Document B.2, this issue would not necessarily occur. Others suggested that it would be better to have the ability to return to that specific point and proceed to the final stage without repeating the entire process. The addition of an overview page with editing possibilities at the end before reaching the final stage would be a solution for this, we will go into more detail on this later in Section 5.5.

In the case of Q13 and Q14, both refer to the pleasantness and enjoyment of using the system interface, having an average score of 2.30 and 2.20 respectively. During the interview, most expressed that the design is very simple and that they would like it to be more interactive and have the images of each device to select from rather than just text.

Finally the last part of the questionnaire is the Informative Questionnaire, the participants answered 3 yes/no questions gathering comments for each of them. First the results of the first question (IQ1) refers to whether the participants found any challenge or complexity in the system, obtaining as answer "No" by 100% of the participants indicating a positive system performance, comments such as "It was straight to the point", "Everything was clear and understandable" were consistent across participants. For the second question (IQ2) which refers to the satisfaction with the FUI of the IoT application generated, 90% of the participants answered "Yes", however when capturing comments about if they would like to improve it, 2 participants indicated that they would like different buttons, another participant wanted the possibility to remove the name of the application in the FUI, then another participant indicated that the FUI generated should be more animated and more pleasant, while the rest of the participants did not leave a response. For the third question (IQ3), we asked if the tool provides enough options for customisation and 60% of the participants answered "Yes". Some specific comments from the participants who answered "No", were "It would be nice to have the possibility to change the font", "Button and font size should be customise", "More icons and images" and "more buttons type for example a slide button to turn on the lights so we can control the intensity and font styles and size".
The last question (IQ4) measures the level of difficulty each participant felt while performing the tasks. The results are shown in Figure 5.2, and it can be seen that all participants found the tasks very easy and easy to complete. This is the outcome that is consistent with IQ1, where participants did not experience any complexity or difficulty during the tasks and left positive feedback on this point as previously mentioned.

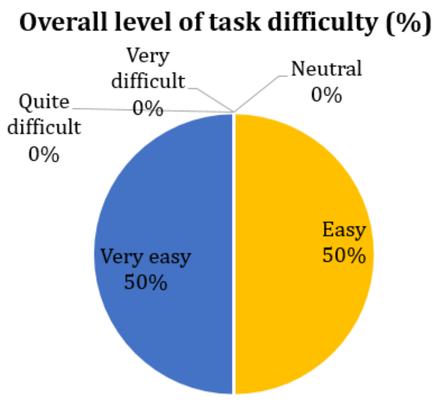


Figure 5.2: Level of difficulty to perform the tasks

5.4.2 Observations

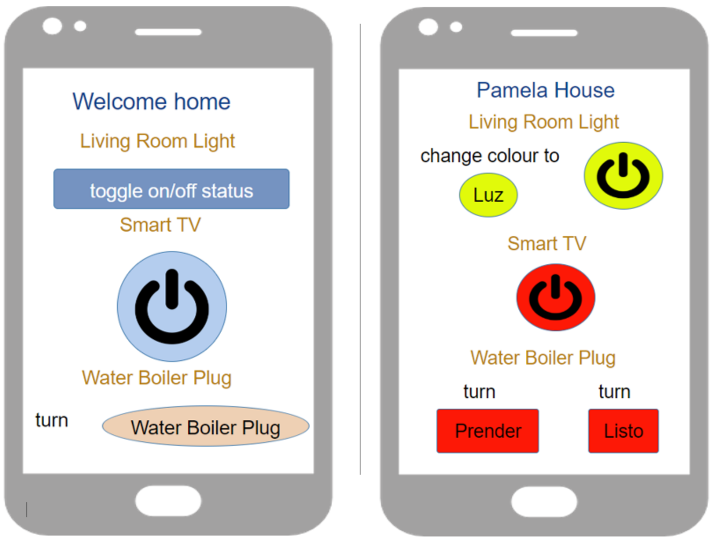
During the user study we observed some results and difficulties from the participants that provide us with more information about various aspects of the system that need to be taken into consideration. To understand these observations, we summarise the steps that the participants performed during the tasks below, but for more details about the steps, see Section 4.1.1:

1. Define the name and description of the application.
2. Select the smart devices to be used in the application for which GUI can be created.
3. Select the smart devices or *things* with the actions you want to control for each smart device selected in previous step.
4. Customise the buttons for each smart device or *thing* selected in previous step if desired or go straight to step 5.
5. Choose your preferred final user interface.

The steps are straightforwards according to the participants, however step 2 was the most complicated step for participants who are not familiar with these IoT authoring tools; most of them were looking for the smart devices and *things* they wanted to control and when they did not find them, they had doubts about what to select and got confused. However, when we asked how to rephrase it for a better understanding, they indicated that they would not change it, that they just needed that context to understand the difference and be able to move on.

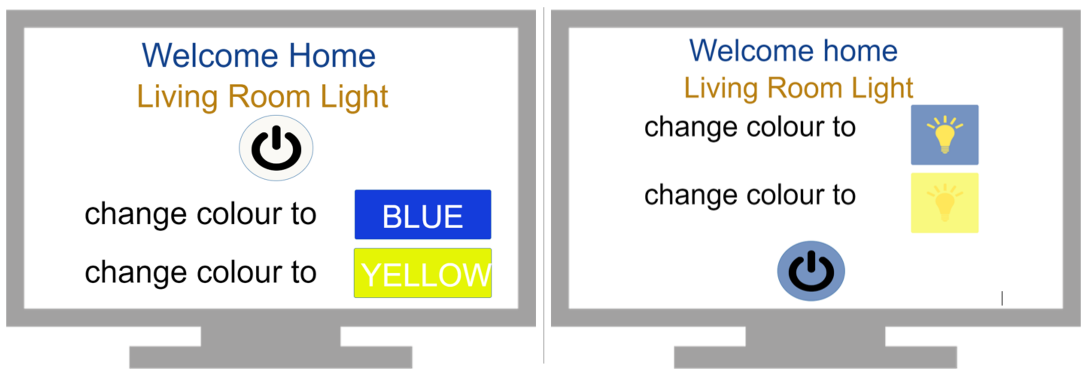
The average amount of time it took the participants to complete the tasks was 29.17 minutes, but there were two factors that affected this, one of them is the fact that some participants took more time to read the Scenario B.1 and the Task Document B.2, while others simply went to the system and started solving the tasks as they read. The other factor we observed was a difference between both gender (female and male) among our participants, the 3 men in our user study abused the "try luck!, continue" button where the system chooses the values for customisation. While the women in our study took their time choosing the colours of the text and button. Therefore, the average time for men in our study was 17.5 minutes, whereas the average time for women was 35 minutes.

Another observation is with respect to the Final User Interfaces (FUI) generated by the participants, for the first application the participants did not know what to expect from the result so the choice of customisation during the process was not always consistent with their previous choices, resulting in FUIs lacking visual appeal, we share some examples in Figure 5.3, we also share some examples in Figure 5.4 of participants with a clear vision of the desired end result and designed their FUIs accordingly. It is important to mention that once the FUI was generated, there was still the possibility to go back and generate the FUI again; however, only one participant was seen to go back and enhance their FUI.



a) Task 1: Outcome participant "P1" b) Task 1: Outcome participant "P5"

Figure 5.3: Poorly designed FUI from the user study



a) Task 1: Outcome participant "P4"   b) Task 1: Outcome participant "P7"

Figure 5.4: Better designed FUI from the user study

One final remark is that during the final stage of the system, the participants were presented with a choice between two preference options (option A/option B). The majority of participants chose option A, as option B was perceived as messy. In Figure 5.3, the right image, provides an example of option B for reference. This observation highlights the preference for elements vertically aligned in the centre.

5.5 Summary and Discussions

We conducted a user study with 10 participants who had little knowledge of design skills or programming. It is important to note that the number of participants is relatively low, so it may not accurately reflect a larger population. Nevertheless, we can still gain insights and identify potential improvements based on how the participants behaved and interacted with the system.

We requested participants to complete the PSSUQ after using the system, and by analysing their responses, we could infer that the results of the user study was relatively positive, with a favourable perception by the participants regarding the usability of the system and the information and interface quality. Participants were also asked to complete the informative questionnaire, which gave us more specific details on how to improve the system, possible new requirements, among other specifications as they were open questions. The most requested feature was to add more customisation options such as the addition of more button types, the possibility to change font and font size and the addition of icons. Additionally from this questionnaire we reaffirmed an observed behaviour, which is that none of the participants encountered any complexity during the use of the system, with the exception of the second step, where participants without experience with home authoring tools were confused about the smart devices on which FUIs can be created and the smart devices and *things* they want to control. However, this exception only occurred in the first task and not in the second task of the Task Document B.2. The same disparity in treatment was noted in the generated FUI. The generated FUI of the first task was sometimes poorly designed by some participants, but when using the tool for the second task, some of them improved their FUI with more consistent colours and buttons selection. This brings us to the next issue, which is that the generated FUI was not always visually appealing. Some participants did not know what to expect after completing all the steps of the process, so their customisation choices were not always consistent with their previous choices. A possible solution for this issue is shown on Figure 5.5, where a preview of the FUI could be provided at all times to allow the end user to have better control over the customisation options. Note that there is a navigation bar at the bottom of the preview container, this is because the system can offer more than one alternative for selecting a preferred FUI during the final stage.

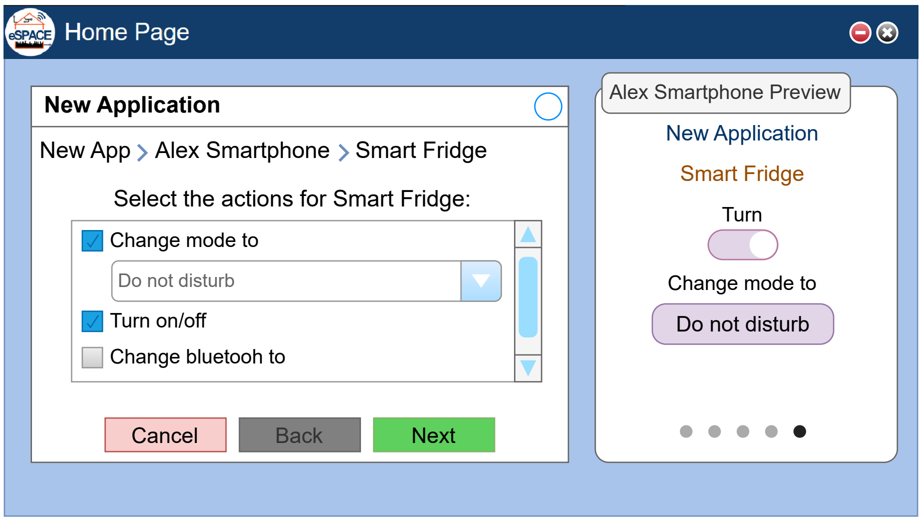
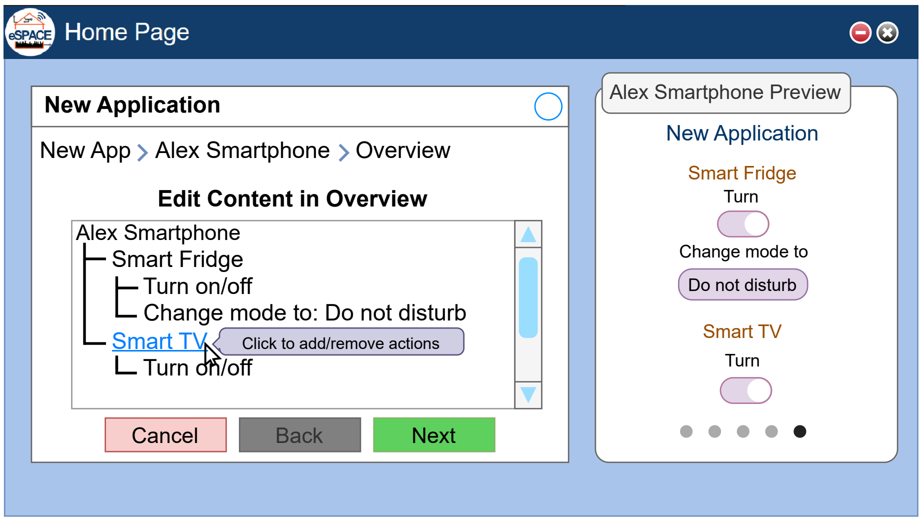


Figure 5.5: Mockup of the "Generate App" option showing the preview

Finally, the last observation was regarding the frustration of the participants with having to repeat the steps in case they forgot to choose a device or action. As a result, a new step can be added at the end, just before the final question about which FUI to select, as shown in Figure 5.6. The overview is shown as a horizontal tree view of the content on the active screen device. ach node is an editable option that directs to that particular step and offers tooltip messages when pointed at to provide more information about what one can change. After editing, the same overview screen should be displayed again as long as one needs to make further changes at that point.



Alex Smartphone
- Smart Fridge
  - Turn on/off
  - Change mode to: Do not disturb
- Smart TV — Click to add/remove actions
  - Turn on/off

Figure 5.6: Mockup of the overview content with editable options

6 Conclusion

In this chapter, we conclude the research of this thesis by summarising and analysing the solution we proposed, to address the main research question and its underlying research sub-questions. We also examine the areas for improvement that were identified during the evaluation of our proof-of-concept prototype.

6.1 Conclusion

In this work we proposed the automatic generation of applications, for controlling IoT devices as an answer to our main research question: *How can we provide a user interface (UI) generation module that allows end users to create their own Internet of Things (IoT) application UIs without any prior programming knowledge or UI design skills?.*

We designed a proof-of-concept prototype based on the derived sub-questions, which led us to the research of studies in the area of automatic user interface generation and the implementation of our solution on top of the eSPACE authoring tool, a semi-automatic tool that provides control over smart devices and *things* on a single platform. Below, we present an overview of our work by responding to each of the research sub-questions.

Research Question 1 (RQ1) *What are the necessary requirements for the end user to generate a user interface without programming or UI design skills?*

First, we identified the mechanisms and techniques required to generate a user interface (UI), by analysing the related work on a number of model-based tools in the area of automatic UI generation in any context. This analysis helped us to determine the requirements for end users without programming or UI design knowledge. We listed these requirements in Table 6.1.

Requirement 1 (R1)	Support for end users to model the specification of the UI.
Requirement 2 (R2)	Generate a quickly and efficient UI design with minimal effort.
Requirement 3 (R3)	Integrate UI design with the application functionality.
Requirement 4 (R4)	Provide a friendly and simple user interface easy to use.
Requirement 4.1 (R4.1)	Simplicity in design.
Requirement 4.2 (R4.2)	The module should be of quality.

Table 6.1: Requirements for end users to generate a UI

The first requirement reflects the need to provide the user with access to a module that can help them gather all the specifications for the user interface they want to generate and addresses our target user, which are users who lack knowledge of programming and/or UI design. The second requirement focuses on the fact that the process that generates the UI must be a fully automatic solution, whose tools aim to simplify the effort of the users, and provide the quickest possible process. It also seeks to ensure that the UI that is developed is efficient, which brings us to our third requirement, the generated UI must be functional. This requirement aligns us to our specific context, which is IoT applications. Therefore the UI must be designed for these applications and must offer a complete user experience.

Finally our last requirement focuses on the UI of the module in which the user must interact to create their IoT application. This UI must have a simple design as our target user can group users with no experience in these applications and as any fully automatic tool, it should be of quality.

Research Question 2 (RQ2) *What are the necessary concepts needed for creating a module that allows end users to generate their own UIs for controlling their smart devices and* things*?*

We previously indicated that instead of starting from scratch, we were going to start from an existing solution. The eSPACE authoring tool is that solution that will allow us to generate applications capable of controlling smart devices and *things*. However, as discussed in our requirements, we want a fully automatic solution, and since the eSPACE authoring tool is a semi-automatic tool, then we need to level it up by extending it with a new fully automatic module. As extending the tool is not an intuitive process, in Chapter 3, we present the eSPACE reference framework that includes all the components and concepts needed to create IoT applications. There was no need to introduce new concepts to design our module.

Research Question 3 (RQ3) *How can we extend an existing UID tool with UI generation options as a result of the requirements and new concepts gained from RQ1 and RQ2?*

We created a new fully automatic module on top of the eSPACE authoring tool in response to this question. The reference framework, which we studied in response to RQ2, helped us model the fluid process of our module to generate IoT applications. As our module should be based on the resulting requirements of RQ1, we then introduced in Section 3.2, additional concepts and guidelines that helped us to design an implement our module in Chapter 4. As a result, our module that extends the tool is a multi-step form wizard capable to gather the user's requirements and preferences and providing them with two different designs for the generated UI at the end. The generation of the IoT final user interface is also functional minimising the effort required by the user to create an IoT application.

After the deployment was complete, we conducted a user study to evaluate our module. This study included three post-questionnaires to get more data from the participants regarding their interactions with the tool and any potential ideas they may have. Despite the fact that we did not offer any training or tutorials on the tool during the evaluation, the satisfaction with the perceived usability of the system according to the PSSUQ is good. All participants agreed that there was no complexity along the way and it was simple to create their applications. However, despite the good results, there are several areas that need to be improved.

For instance, we could observe that our current module does not always generate a well-designed FUI, and also the participants agreed that they would like more customisation options for the final user interface. In the next section, we will discuss and provide more information on these points and our suggestions for improvements.

6.2 Future Work

There were a few shortcomings found during the implementation and evaluation of the module that should be taken into account for future work. The first area of improvement that needs to be taken into account is making sure that the default templates of the generated final user interface options are always distinct from one another following the basic principles of UI design, which is not always the case in our prototype. Given that this problem arises when there are few elements on the FUI, then in a future version we should add additional rules, that considers a different alignment with respect to the device name label and its buttons or other UI element, as shown in Figure 4.16.

Another area for improvement is to ensure that we always provide a well designed FUI, although this last point depends on the end user, our prototype can still facilitate this task by providing a preview of the FUI during the entire process, Figure 5.5 is a mockup of how such a solution can be implemented, which will give the user more control over their customisation options. In addition, we should also implement the recommendations and requests of the users, some of them stated that they would like more customisation options. The next step would therefore be to enable users to change the font and font size as well as add more options for other representations used to execute the actions, such as sliders, icons, toggle switches, checkboxes, among others. We should also add icons next to the corresponding device name to make it easier and quicker for users to find and select the devices they want to control. Further, in response to another request in wanting to redirect to the exact point to edit some selection without having to go back and redo all the steps again, in Figure 5.6, we present an overview option with editable options is presented as a solution to this problem, so that only the editing of that element is allowed and all previous settings are saved.

Finally, to make our IoT applications more powerful, we should also add the complementary elements of the eSPACE authoring tool that were not considered in this initial phase due to time constraints, such as the following widgets:

- Weather: We can add the current weather at the location of the screen device where the application is located. This is a simple UI element that can be displayed on the FUI.
- Video player: This item must be able to support a full screen and also be able to display static, live and streaming videos. This requires two new UI elements in the FUI: a video frame (UIe) linked to a video stream and a button (UIe) linked to an AC that can capture videos.
- Lists: As an IoT application can consist of multiple FUIs for different devices, a list can be present in two FUIs and be synchronised across the two FUIs. Therefore, if the list is edited on one of the devices, these changes are also made on the next device.

It is important to note that we should add an additional step in the current process of our module. This step would involve asking the user whether they would like to incorporate widgets. Depending on the chosen widget, the module should request the user for the necessary settings. Additionally, we might need to add new design rules in the default FUI template to accommodate these new items and perform unit tests to check that a well designed FUI is preserved.

Appendix A

Mockups

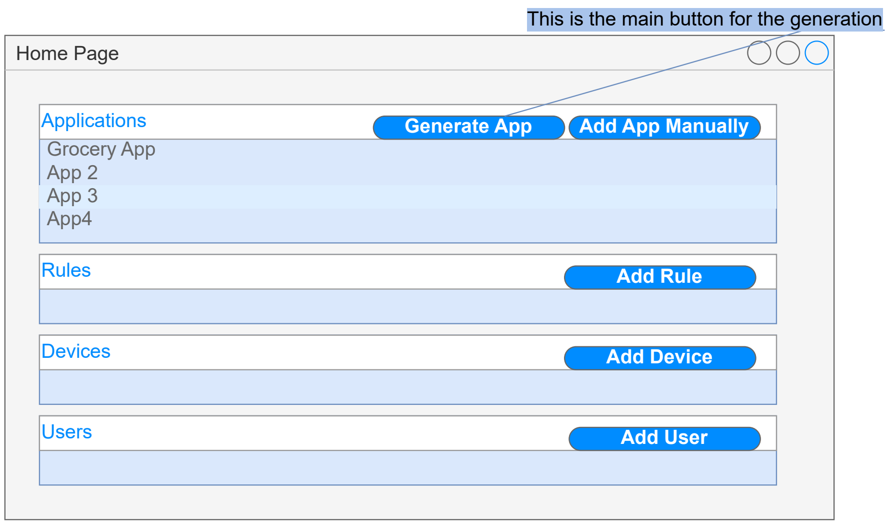


Figure A.1: Generate IoT Application - Mockup Part 1

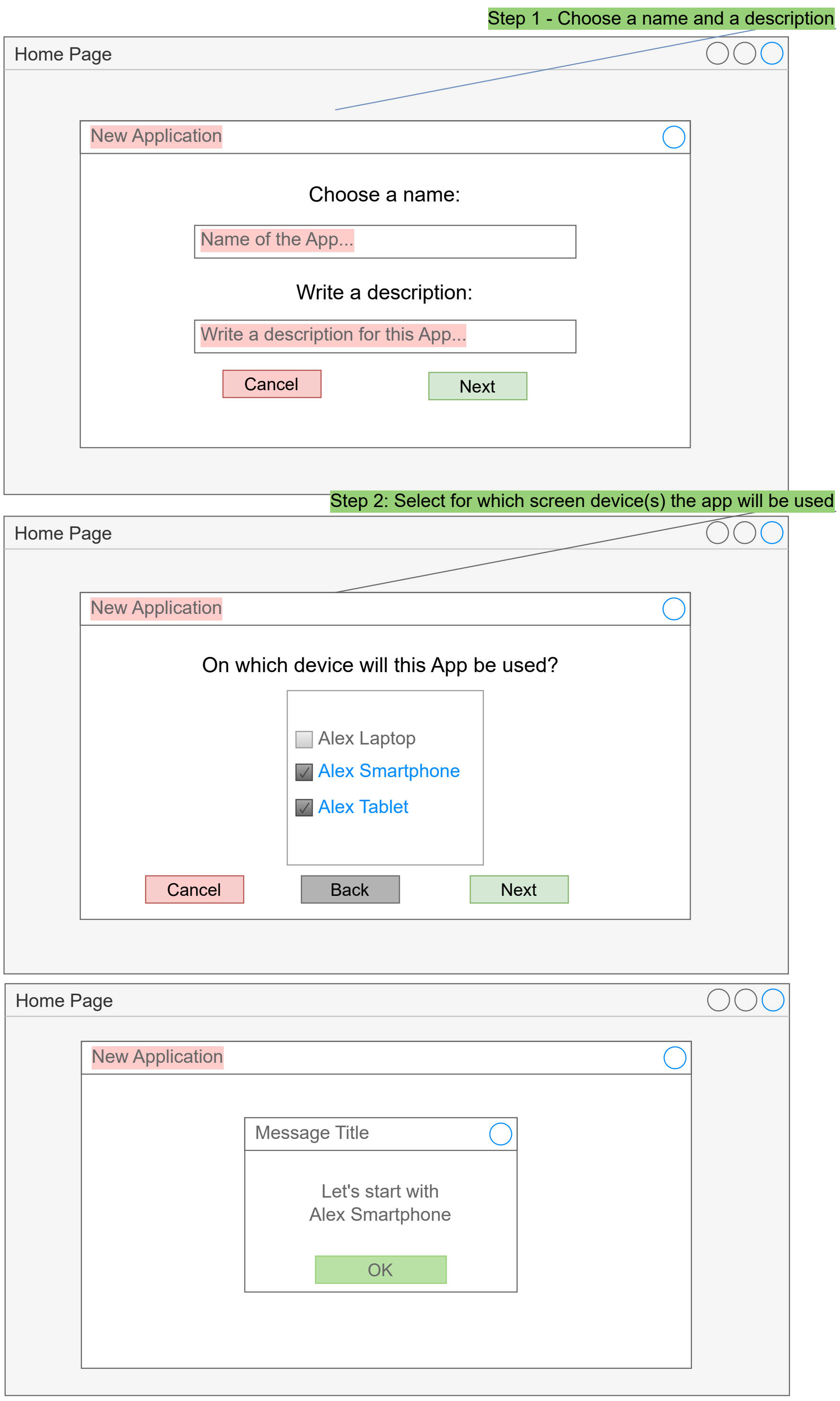


Figure A.2: Generate IoT Application - Mockup Part 2

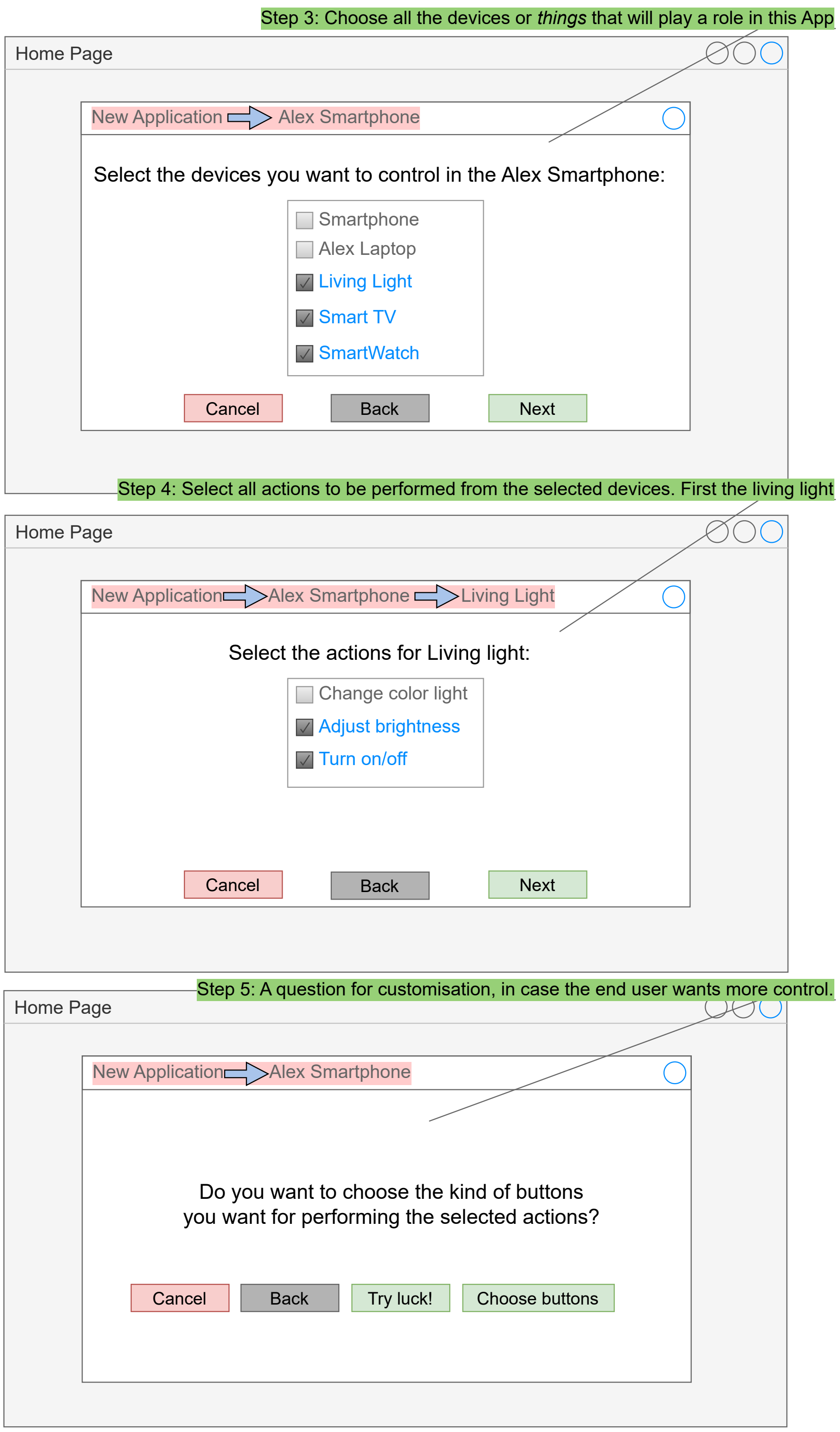
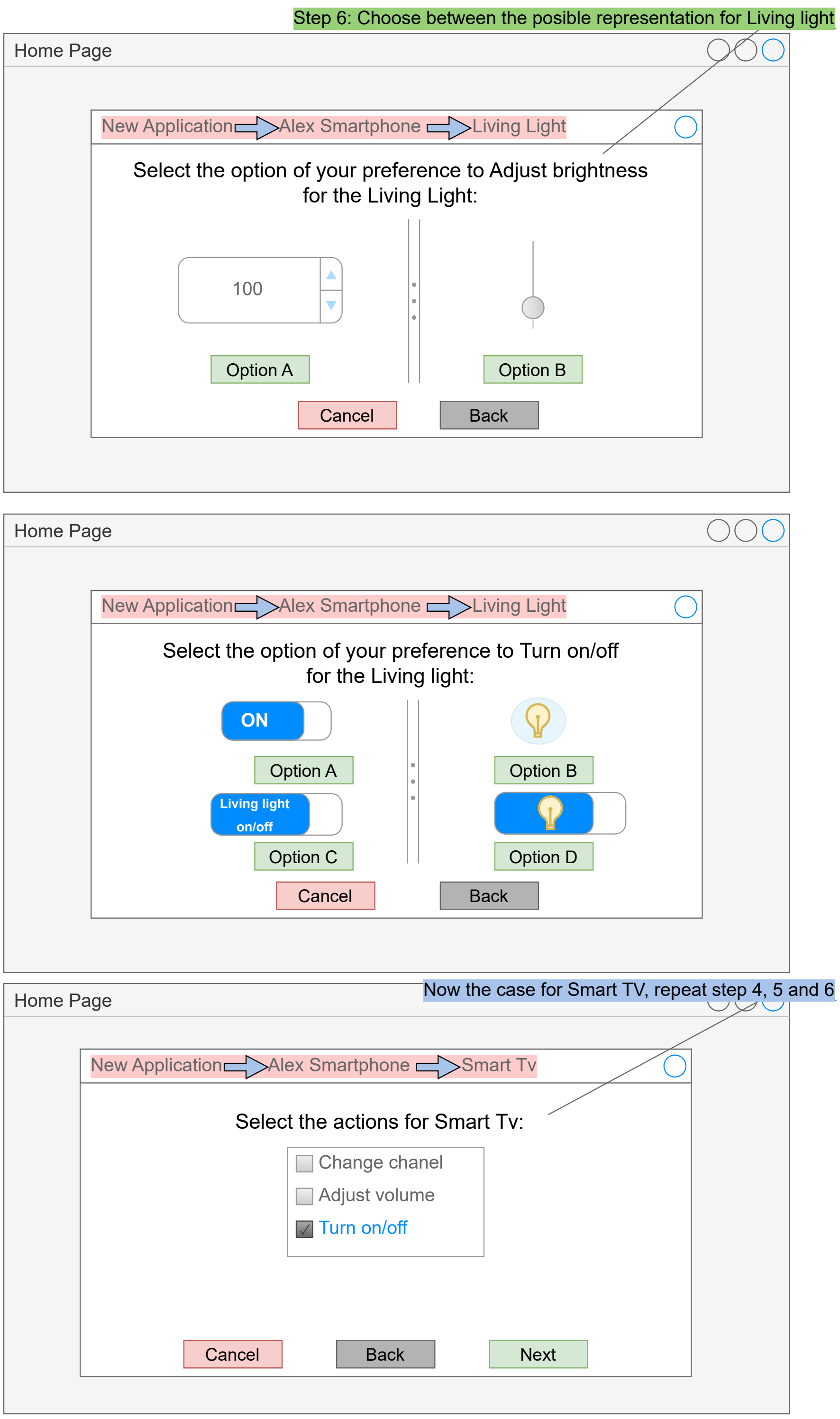

Step 3: Choose all the devices or *things* that will play a role in this App
Home Page
New Application
Alex Smartphone
Select the devices you want to control in the Alex Smartphone:
Smartphone
Alex Laptop
Living Light
Smart TV
SmartWatch
Cancel
Back
Next
Step 4: Select all actions to be performed from the selected devices. First the living light
Home Page
New Application
Alex Smartphone
Living Light
Select the actions for Living light:
Change color light
Adjust brightness
Turn on/off
Cancel
Back
Next
Step 5: A question for customisation, in case the end user wants more control.
Home Page
New Application
Alex Smartphone
Do you want to choose the kind of buttons you want for performing the selected actions?
Cancel
Back
Try luck!
Choose buttons

Figure A.3: Generate IoT Application - Mockup Part 3


Step 6: Choose between the posible representation for Living light

Home Page

New Application → Alex Smartphone → Living Light

Select the option of your preference to Adjust brightness for the Living Light:

Home Page

New Application → Alex Smartphone → Living Light

Select the option of your preference to Turn on/off for the Living light:

Now the case for Smart TV, repeat step 4, 5 and 6

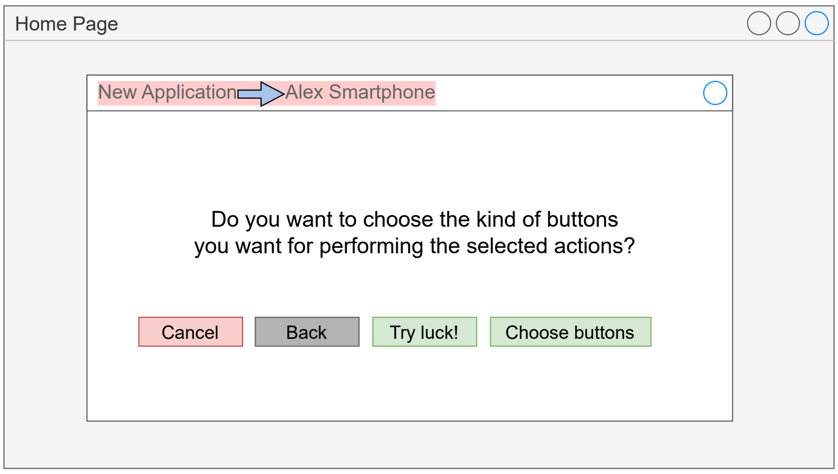
Home Page

New Application → Alex Smartphone → Smart Tv

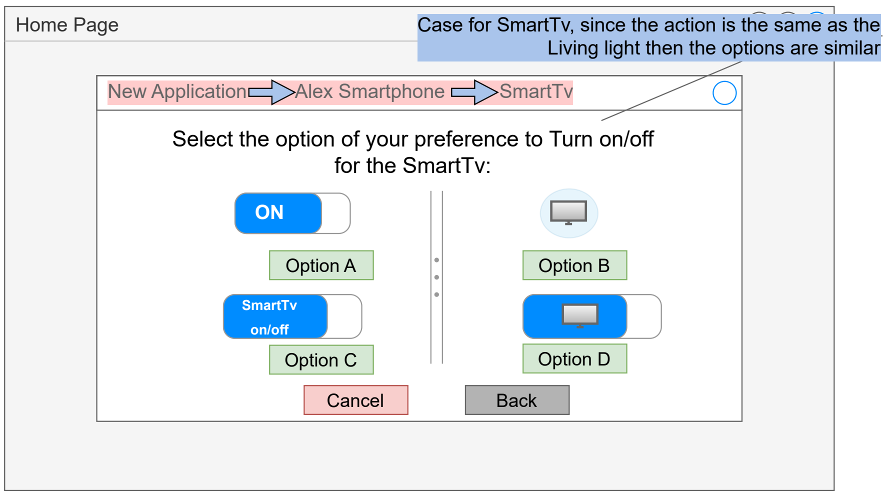
Select the actions for Smart Tv:

- [ ] Change chanel
- [ ] Adjust volume
- [x] Turn on/off

Figure A.4: Generate IoT Application - Mockup Part 4



Do you want to choose the kind of buttons you want for performing the selected actions?



Case for SmartTv, since the action is the same as the Living light then the options are similar

Select the option of your preference to Turn on/off for the SmartTv:

Steps 4 to 6 are repeated for the next device: SmartWatch

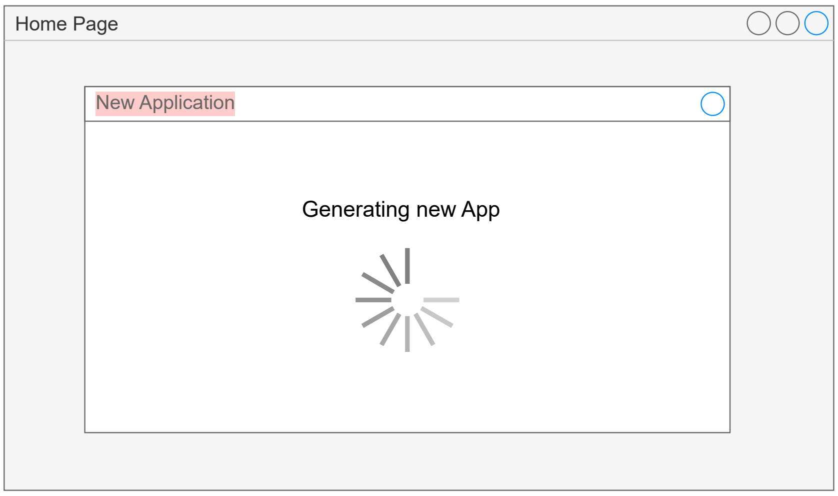
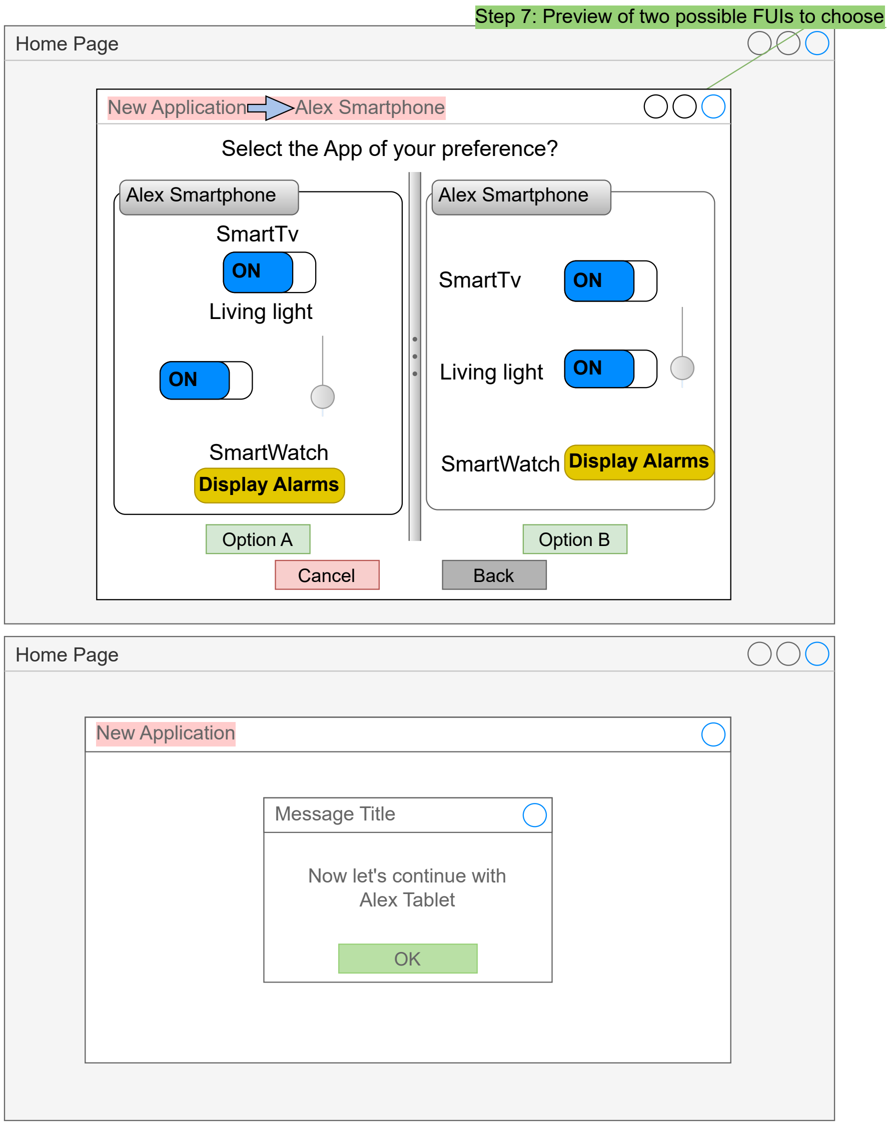


Figure A.5: Generate IoT Application - Mockup Part 5



Steps 3 to 7 are repeated for the next device: Alex Tablet

Figure A.6: Generate IoT Application - Mockup Part 6

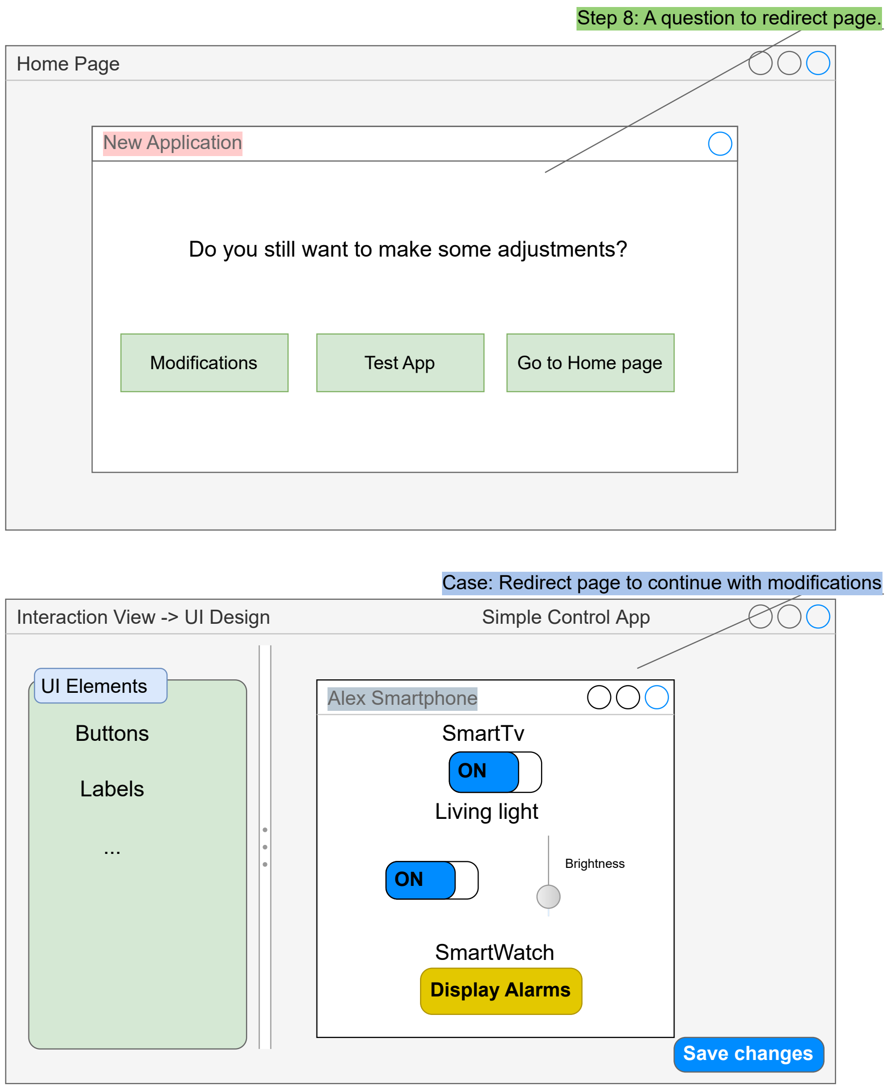


Figure A.7: Generate IoT Application - Mockup Part 7

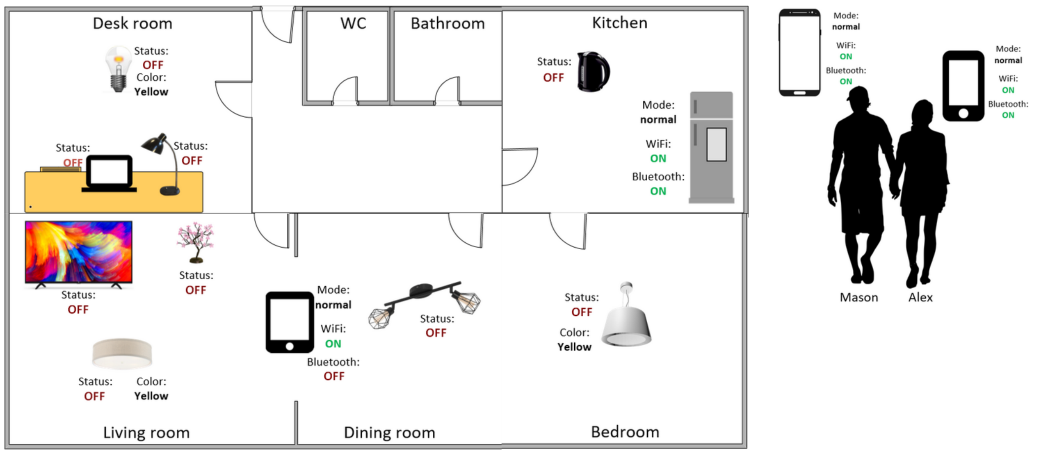
B

Appendix B

Evaluation

B.1 Scenario

In this studio you will impersonate Alex, a young student who wants to create applications to control her Smart devices in her home that resemble her routines. Below you can see the plan of her Smart home:



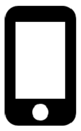
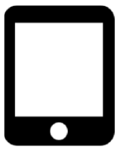
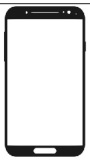
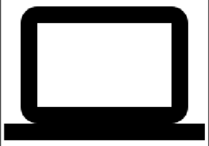
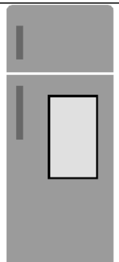
As can be seen the Smart home plan contains the following Internet of Things (IoT) devices such as 4 smart lights, as shown below:

Living Room Light	Bedroom Light	Desk Light	Cat Room Light

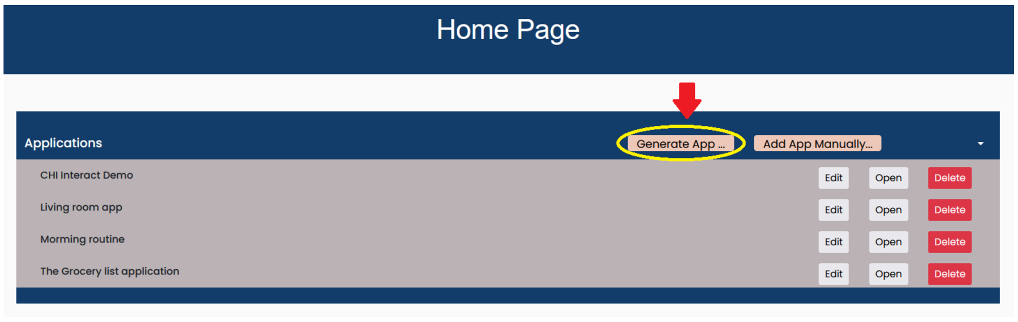
2 smart power plugs, as shown below:

Water Boiler Plug	Light Tree Plug

6 screen devices for which graphical user interfaces (UI) can be created.

 Mason Smartphone	 Living Room Tablet	 Alex Smartphone
 Alex Laptop	Smart TV	 Smart Fridge

The tool you will use to control and interact with the Smart devices and Things is the eSPACE authoring tool. Instead of manually building the applications for these particular tasks, you will use the "Generate App..." option to create the applications, as shown below:



Home Page

Applications — Generate App ... — Add App Manually...

CHI Interact Demo	Edit	Open	Delete
Living room app	Edit	Open	Delete
Morning routine	Edit	Open	Delete
The Grocery list application	Edit	Open	Delete

B.2 Task Document

User Study

For the following study you will have the opportunity to complete two tasks using the eSPACE authoring tool, each task is intended to simulate Alex's daily routines. To make it easier to follow along, we use the color **blue** for Alex's smart devices and **red** for the actions to be executed.
eSPACE authoring tool gives the possibility to choose custom buttons, we encourage you to make use of the various options available to you. Once you have created your application, it is recommended that you try it out and see its functionality.
If you have any questions, please don't hesitate to reach out to your supervisor, Brenda Ordonez.

Task 1: Home Arrival Application

Imagine you are Alex, and you want to make the following application called "Welcome Home". For this application you want to create two designs (user interfaces), one for **Alex Smartphone** and the other one for the **Smart TV**.
For **Alex Smartphone**, the following actions should be possible to be executed:

- Turn **on** the **Living Room Light**.
- Turn **on** the **Water Boiler Plug**.
- Control over the **Smart TV** (**on/off**).

For the **Smart TV**, the following actions should be possible to be executed:

- Control the colour and status of the **Living Room Light** by allowing Alex to **change the colour** to blue, yellow and allow her to turn it **on** or **off**.

Task 2: Bedtime Routine Application

Imagine you are Alex, and you want to make the following application called "Bedtime routine", where the following actions should be possible to be executed on her **smartphone**:

- Control of the lights (**Cat Room Light, Living Room Light, Bedroom Light**) to perform the **on/off** action for each light.
- Control of the **Smart Fridge** to **change the mode** to not disturb.

B.3 Post-Survey Questionnaires

User Study Informed Consent Form

Study administrator is: Brenda Ordonez
Participant is:____________________
Participant number:________________

This is a study on the generation of IoT applications. Our goal is to facilitate end users in creating customized IoT applications for a smart home environment based on their specific requirements and preferences. Your participation will help us achieve this goal.
In this session you will be working with the eSPACE authoring tool which is running in the web browser. We will ask you to perform tasks a typical user might do, described in the scenario that you will receive. The study administrator will sit in the same room, quietly observing and taking notes. This person will sit near you and help you if you are stuck or have questions. All information collected in the session belongs to the VUB WISE lab and will be used for internal purposes. We will record your screen and voice during the session. We may publish our results from this and other sessions in our reports, but all such reports will be anonymised and will not include your name.
We are not testing you. We want to find out what aspects need improvements, so we can make it better. You may take breaks as needed and stop your participation in the study at any time.

Statement of Informed Consent

I have read the description of the study and of my rights as a participant. I voluntarily agree to participate in the study.

Print Name:________________
Signature:__________________
Date:______________________

B.3.1 Personal Information

The following questionnaire has the purpose of gaining insight of the people on this user study created in the context of the master thesis: "Investigation of the generation of IoT application". Please fill with your personal information.

Age:____________________ Gender: ____________________

Highest degree obtained:

- ☐ High school diploma
- ☐ Bachelor degree
- ☐ Master degree
- ☐ Ph.D
- ☐ Other: ________

Education: ____________________

Do you have experience with Smart Home applications or similar Internet of Things (IoT) applications? Yes| No.
If yes, please indicate which ones.

- ☐ Amazon Alexa
- ☐ Google Home
- ☐ IFTTT
- ☐ Philips Hue
- ☐ Samsung SmartThings
- ☐ Other: ________

B.3.2 Post-Study System Usability Questionnaire (PSSUQ)

Strongly agree Strongly disagree

1	2	3	4	5	6	7	N/A

On a scale between Strongly Agree to Strongly Disagree as shown above, please rate the following statements:

1. Overall, I am satisfied with how easy it is to use this system.
2. It was simple to use this system.
3. I was able to complete the tasks and scenarios quickly using this system.
4. I felt comfortable using this system.
5. It was easy to learn to use this system.
6. I believe I could become productive quickly using this system.
7. The system gave error messages that clearly told me how to fix problems. **(not considered)**
8. Whenever I made a mistake using the system, I could recover easily and quickly.
9. The information (such as online help, on-screen messages, and other documentation) provided with this system was clear. **(not considered)**
10. It was easy to find the information I needed.
11. The information was effective in helping me complete the tasks and scenarios.
12. The organisation of information on the system screens was clear.
13. The interface of this system was pleasant.
14. I liked using the interface of this system.
15. This system has all the functions and capabilities I expect it to have.
16. Overall, I am satisfied with this system.

B.3.3 Post-Study Informative Questionnaire

Please complete the final set of questions.

1. Did you encounter any challenge or complexity using the tool?
   Yes | No.
   Can you give more details on your answer? ____________________
   ____

2. In general, are you satisfy with the generated app designs (UI)?
   Yes | No.
   is there anything you would like to improve in the generated app design (UI)?
   ____

3. Did the tool provide enough options for customisation?
   Yes | No.
   If no, please provide more details about what additional customisation options you feel are necessary?______________________________
   ____

In general, how did you find this tasks?

Very difficult(1)	Quite difficult(2)	Neutral(3)	Easy(4)	Very easy(5)

Thanks for your participation!

C
Appendix C

Evaluation Results

Participant: P1 - 07/07/2023

Personal Information
Age: 30
Gender: Female
Highest degree obtained: Bachelor degree
Education: Business administration
Experience with Smart Home applications or similar Internet of Things (IoT) applications: No
PSSUQ
Q1: 4
Q2: 2
Q3: 3
Q4: 1
Q5: 1
Q6: 1
Q7: 4
Q8: 1
Q10: 1

Q11: 1
Q12: 1
Q13: 1
Q14: 1
Q15: 2
Q16: 1
Overall PSSUQ: 1.50
SYSUSE: 2.00
INFOQUAL: 1.00
INTERQUAL: 1.33
Post-Study Informative Questionnaire
IQ1: No. Comments: Everything was clear and understandable
IQ2: No. Comments: At the final design, the title is not on the center, looked messy.
IQ3: No. Comments: I didn't prove the customisation tool
IQ4: Easy

Participant: P2 - 07/07/2023

Personal Information
Age: 33
Gender: Male
Highest degree obtained: Master degree
Education: Msc in Management
Experience with Smart Home applications or similar Internet of Things (IoT) applications: No
PSSUQ
Q1: 2
Q2: 2
Q3: 1
Q4: 1
Q5: 2
Q6: 1
Q7: 3
Q8: 3
Q10: 2
Q11: 1
Q12: 2
Q13: 3
Q14: 3
Q15: 2

Q16: 2
Overall PSSUQ: 1.93
SYSUSE: 1.50
INFOQUAL: 2.00
INTERQUAL: 2.67
Post-Study Informative Questionnaire
IQ1: No. Comments: It was straight to the point
IQ2: Yes. Comments: Easy to use and to test
IQ3: No. Comments: It would be nice to have the possibility to change the font
IQ4: Very easy

Participant: P3 - 07/07/2023

Personal Information
Age: 34
Gender: Female
Highest degree obtained: Master degree
Education: MSc in Chemistry
Experience with Smart Home applications or similar Internet of Things (IoT) applications: Yes, Samsung SmartThings
PSSUQ
Q1: 2
Q2: 2
Q3: 3
Q4: 3
Q5: 2
Q6: 1
Q7: 3
Q8: 2
Q10: 2
Q11: 2
Q12: 1
Q13: 3
Q14: 3
Q15: 2
Q16: 2
Overall PSSUQ: 2.14
SYSUSE: 2.17
INFOQUAL: 1.75
INTERQUAL: 2.67

Post-Study Informative Questionnaire
IQ1: No. Comments: The steps were very clear and the interface was intuitive
IQ2: Yes. Comments: I would like more button options to choose from.
IQ3: No. Comments: Button and font size should be customizable.
IQ4: Easy

Participant: P4 - 09/07/2023

Personal Information
Age:29
Gender: Female
Highest degree obtained: Bachelor degree
Education: Industrial Engineer
Experience with Smart Home applications or similar Internet of Things (IoT) applications: Yes, Amazon Alexa
PSSUQ
Q1: 1
Q2: 1
Q3: 1
Q4: 1
Q5: 2
Q6: 1
Q7: 1
Q8: 3
Q10: 1
Q11: 1
Q12: 1
Q13: 1
Q14: 1
Q15: 1
Q16: 1
Overall PSSUQ: 1.21
SYSUSE: 1.17
INFOQUAL: 1.50
INTERQUAL: 1.00
Post-Study Informative Questionnaire
IQ1: No. Comments: The interfaces were easy to understand according to your requirements
IQ2: Yes. Comments: -
IQ3: Yes. Comments: -

IQ4: Very easy

Participant: P5 - 09/07/2023

Personal Information
Age: 39
Gender: Female
Highest degree obtained: Technician
Education: Administrator
Experience with Smart Home applications or similar Internet of Things (IoT) applications: Yes, Amazon Alexa and Samsung SmartThings
PSSUQ
Q1: 1
Q2: 1
Q3: 2
Q4: 1
Q5: 1
Q6: 1
Q7: 1
Q8: 1
Q10: 1
Q11: 1
Q12: 1
Q13: 2
Q14: 2
Q15: 2
Q16: 1
Overall PSSUQ: 1.29
SYSUSE: 1.17
INFOQUAL: 1.00
INTERQUAL: 2.00
Post-Study Informative Questionnaire
IQ1: No. Comments: once the idea was understood, it became easier
IQ2: Yes. Comments: once in test you could modify a specific action without having to go back one step at a time.
IQ3: Yes. Comments: -
IQ4: Easy

Participant: P6 - 10/07/2023

Personal Information
Age: 58

Gender: Male
Highest degree obtained: Ph.D
Education: Electronic Engineer
Experience with Smart Home applications or similar Internet of Things (IoT) applications: No
PSSUQ
Q1: 1
Q2: 1
Q3: 1
Q4: 1
Q5: 1
Q6: 1
Q7: 2
Q8: 1
Q10: 1
Q11: 1
Q12: 1
Q13: 1
Q14: 1
Q15: 1
Q16: 1
Overall PSSUQ: 1.00
SYSUSE: 1.00
INFOQUAL: 1.00
INTERQUAL: 1.00
Post-Study Informative Questionnaire
IQ1: No. Comments: all clear
IQ2: Yes. Comments: No
IQ3: Yes. Comments: -
IQ4: Easy.

Participant: P7 - 10/07/2023

Personal Information
Age: 29
Gender: Female
Highest degree obtained: Bachelor degree
Education: Mining engineer
Experience with Smart Home applications or similar Internet of Things (IoT) applications: Yes, Google Home
PSSUQ

Q1: 1
Q2: 1
Q3: 1
Q4: 2
Q5: 1
Q6: 1
Q7: 4
Q8: 4
Q10: 1
Q11: 1
Q12: 1
Q13: 4
Q14: 4
Q15: 3
Q16: 2

Overall PSSUQ: 1.93
SYSUSE: 1.17
INFOQUAL: 1.75
INTERQUAL: 3.67
Post-Study Informative Questionnaire
IQ1: No. Comments: -
IQ2: Yes. Comments: The interface to be more animated or more pleasing to the eye
IQ3: No. Comments: More icons, images
IQ4: Very easy

Participant: P8 - 11/07/2023

Personal Information
Age: 27
Gender: Female
Highest degree obtained: Bachelor degree
Education: Biotechnological engineer
Experience with Smart Home applications or similar Internet of Things (IoT) applications: Yes, Amazon Alexa
PSSUQ
Q1: 2
Q2: 2
Q3: 3
Q4: 3

Q5: 2
Q6: 4
Q7: 4
Q8: 3
Q10: 2
Q11: 2
Q12: 1
Q13: 2
Q14: 3
Q15: 2
Q16: 2
Overall PSSUQ: 2.36
SYSUSE: 2.67
INFOQUAL: 2.00
INTERQUAL: 2.33
Post-Study Informative Questionnaire
IQ1: No. Comments: It was easy to follow along, i just got confused trying to accomplish alex's instructions
IQ2: Yes. Comments: Maybe have the possibility to choose if we want to display the name of devices
IQ3: No. Comments: More buttons type for example a slide button to turn on the lights so we can control the intensity and font styles and size
IQ4: Easy

Participant: P9 - 11/07/2023

Personal Information
Age: 30
Gender: Female
Highest degree obtained: Bachelor degree
Education: Industrial Engineer
Experience with Smart Home applications or similar Internet of Things (IoT) applications: Yes, Samsung SmartThings
PSSUQ
Q1: 2
Q2: 1
Q3: 2
Q4: 2
Q5: 1
Q6: 1
Q7: 4

Q8: 1
Q10: 1
Q11: 1
Q12: 2
Q13: 3
Q14: 2
Q15: 1
Q16: 1
Overall PSSUQ: 1.50
SYSUSE: 1.50
INFOQUAL: 1.25
INTERQUAL: 2.00
Post-Study Informative Questionnaire
IQ1: No. Comments: Every step is straightforward
IQ2: Yes. Comments: I would like more options to choose from, for the final app
IQ3: Comments: Font style, more buttons
IQ4: Very easy

Participant: P10 - 11/07/2023

Personal Information
Age: 42
Gender: Male
Highest degree obtained: Bachelor degree
Education: Law
Experience with Smart Home applications or similar Internet of Things (IoT) applications: Yes, Amazon Alexa and Samsung SmartThings
PSSUQ
Q1: 1
Q2: 1
Q3: 2
Q4: 1
Q5: 1
Q6: 3
Q7: 4
Q8: 5
Q10: 1
Q11: 2
Q12: 1
Q13: 3

Q14: 2
Q15: 1
Q16: 2
Overall PSSUQ: 1.86
SYSUSE: 1.50
INFOQUAL: 2.25
INTERQUAL: 2.00
Post-Study Informative Questionnaire
IQ1: No. Comments: Didn't find any.
IQ2: Yes. Comments: no.
IQ3: Yes. Comments: -
IQ4: Very easy

Bibliography

[1] Hammad Ali. Helping End Users Design Better UIs, 2022. `https://wise.vub.ac.be/sites/default/files/theses/ThesisHammadAli.pdf`.

[2] François Bodart, Anne-Marie Hennebert, Jean-Marie Leheureuxa, Isabelle Provot, Benoît Sacré, and Jean Vanderdonckt. Towards a Systematic Building of Software Architectures: the TRIDENT Methodological Guide. In *Proceedings of the Eurographics Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS 1995)*, pages 262–278, Toulouse, France, January 1995.

[3] Letizia Bollini. Beautiful Interfaces. From User Experience to User Interface Design. *The Design Journal*, 20(sup1):S89–S101, September 2017. `https://doi.org/10.1080/14606925.2017.1352649`.

[4] Sketch B.V. Glow-Up: 10 New Sketch Features You Might've Missed in 2022. `https://www.sketch.com/blog/new-features-2022/`, December 2022. Accessed: 2023-02-15.

[5] Woorank by Bridgeline. How Does Mobile Font Size Impact SEO? `https://www.woorank.com/en/edu/seo-guides/mobile-font-size`, 2023. Accessed: 2023-06-22.

[6] Interaction Design Foundation. User Interface (UI) Design. `https://www.interaction-design.org/literature/topics/ui-design`, 2023. Accessed: 2023-06-22.

[7] Krzysztof Gajos and Daniel S. Weld. SUPPLE: Automatically Generating User Interfaces. In *Proceedings of the 9th International Conference on Intelligent User Interface (IUI 2004)*, pages 93–100, Madeira, Portugal, January 2004. `http://dx.doi.org/10.1145/964442.964461`.

[8] Krzysztof Gajos and Daniel S. Weld. Preference Elicitation for Interface Optimization. In *Proceedings of the 18th Annual ACM Symposium*

on User Interface Software and Technology (UIST 2005), pages 173–182, Washington, USA, October 2005. https://doi.org/10.1145/1095034.1095063.

[9] Krzysztof Z. Gajos, Daniel S. Weld, and Jacob O. Wobbrock. Automatically Generating Personalized User Interfaces with Supple. *Artificial Intelligence*, 174(12–13):910–950, August 2010. https://doi.org/10.1016/j.artint.2010.05.005.

[10] Krzysztof Z. Gajos, Jacob O. Wobbrock, and Daniel S. Weld. Automatically Generating User Interfaces Adapted to Users' Motor and Vision Capabilities. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST 2007)*, pages 231–240, Rhode Island, USA, October 2007. https://doi.org/10.1145/1294211.1294253.

[11] Axure Inc. Getting Started with Axure RP. https://docs.axure.com/axure-rp/reference/getting-started-video/, 2023. Accessed: 2023-02-15.

[12] Figma Inc. Designing in the Cloud with Confidence. https://www.figma.com/blog/designing-in-the-cloud-with-confidence/, September 2022. Accessed: 2023-02-15.

[13] The Apache Software Inc. Apache NetBeans 16 Released. https://netbeans.apache.org/blogs/entry/announce-apache-netbeans-16-released.html, December 2022. Accessed: 2023-02-15.

[14] Reyes Juárez-Ramírez, Carlos Huertas, and Sergio Inzunza. Automated Generation of User-Interface Prototypes Based on Controlled Natural Language Description. In *Proceedings of the 38th International Computer Software and Applications Conference Workshops (COMPSACW 2014)*, pages 246–251, Vasteras, Sweden, July 2014. https://doi.org/10.1109/COMPSACW.2014.44.

[15] Erik D. Kennedy. The Responsive Website Font Size Guidelines. https://www.learnui.design/blog/mobile-desktop-website-font-size-guidelines.html, August 2021. Accessed: 2023-06-22.

[16] Maria Kosukhina. IntelliJ IDEA 2022.3 Is Out! https://blog.jetbrains.com/idea/2022/11/intellij-idea-2022-3/, November 2022. Accessed: 2023-02-15.

[17] Aaron Marcus. Dare We Define User-Interface Design? *Interactions*, 9(5):19–24, 2002. `https://doi.org/10.1145/566981.566992`.

[18] Aaron Marcus and Andries Van Dam. User-Interface Developments for The Nineties. *Computer*, 24(9):49–57, September 1991. `https://doi.org/10.1109/2.84899`.

[19] Fan Mo and Jia Zhou. The Influence of Menu Structure and Layout on Usability of Smartwatches. *International Journal of Mobile Human Computer Interaction (IJMHCI)*, 10(1):1–22, January 2018. `https://doi.org/10.4018/IJMHCI.2018010101`.

[20] Brigit Murtaugh. Remote Development Even Better. `https://code.visualstudio.com/blogs/2022/12/07/remote-even-better`, 2022. Accessed: 2023-02-15.

[21] Brad A. Myers. User Interface Software Tools. *ACM Transactions on Computer-Human Interaction*, 2(1):64–103, March 1995. `https://doi.org/10.1145/200968.200971`.

[22] Brad Allan Myers, Scott Hudson, and Randy F. Pausch. Past, Present, and Future of User Interface Software Tools. *ACM Transactions on Computer-Human Interaction*, 7(1):3–28, March 2000. `https://doi.org/10.1145/344949.344959`.

[23] Jeffrey Nichols, Duen Horng Chau, and Brad A. Myers. Demonstrating the Viability of Automatically Generated User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2007)*, pages 1283–1292, California, USA, April 2007. `https://doi.org/10.1145/1240624.1240819`.

[24] Jeffrey Nichols and Andrew Faulring. Automatic Interface Generation and Future User Interface Tools. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2005)*, Oregon, USA, April 2005.

[25] Jeffrey Nichols, Brad A. Myers, and Kevin Litwack. Improving Automatic Interface Generation with Smart Templates. In *Proceedings of the 9th International Conference on Intelligent User Interface (IUI 2004)*, pages 286–288, Madeira, Portugal, January 2004. `https://dl.acm.org/doi/10.1145/964442.964507`.

[26] Jeffrey Nichols, Brad A. Myers, and Brandon Rothrock. UNIFORM: Automatically Generating Consistent Remote Control User Interfaces.

In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2006)*, pages 611—620, Québec, Canada, April 2006. `https://doi.org/10.1145/1124772.1124865`.

[27] Jeffrey Nichols, Brad Allan Myers, Michael C. Higgins, Joseph Hughes, Thomas K. Harris, Ronald Rosenfeld, and Mathilde Pignol. Generating Remote Control Interfaces for Complex Appliances. In *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology (UIST 2002)*, pages 161–170, Paris, France, October 2002. `https://doi.org/10.1145/571985.572008`.

[28] Jeffrey Nichols, Brandon Rothrock, Duen Horng Chau, and Brad A. Myers. Huddle: Automatically generating interfaces for systems of multiple connected appliances. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST 2006)*, pages 279–288, Montreux, Switzerland, October 2006. `https://doi.org/10.1145/1166253.1166298`.

[29] Dan R. Olsen, Sean Jefferies, Travis Nielsen, William Moyes, and Paul Fredrickson. Cross-Modal Interaction Using XWeb. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology (UIST 2000)*, pages 191–200, California, USA, November 2000. `https://dl.acm.org/doi/10.1145/354401.354764`.

[30] Dan Reed Olsen. A Programming Language Basis for User Interface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 1989)*, pages 171–176, Texas, USA, March 1989. `https://doi.org/10.1145/67449.67485`.

[31] Randall Packer and Ken Jordan. *Multimedia: From Wagner to Virtual Reality.* Norton paperback, December 2002.

[32] Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3):45–77, January 2007. `https://doi.org/10.2753/MIS0742-1222240302`.

[33] Emmanouil Perakakis and Gheorghita Ghinea. Responsive Web Design for the Internet Connected TV: The Answer to More Smart TV Content? In *Proceedings of the 5th International Conference on Consumer Electronics (ICCE 2015)*, pages 38–42, Berlin, Germany, September 2015.

[34] Angel R. Puerta. A Model-based Interface Development Environment. *IEEE Software*, 14(04):40–47, June 1997.

[35] Angel R. Puerta, Henrik Eriksson, John H. Gennari, and Mark A. Musen. Model-based Automated Generation of User Interfaces. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1996)*, pages 508–515, Washington, USA, June 1996. `https://dl.acm.org/doi/10.5555/286013.286047`.

[36] Audrey Sanctorum. *Conceptual Foundations for End-User Authoring of Cross-Device and Internet of Things Applications*. PhD thesis, Vrije Universiteit Brussel, 2020. `https://wise.vub.ac.be/sites/default/files/theses/PhDThesisAudreySanctorum.pdf`.

[37] Jeff Sauro and James R. Lewis. *Quantifying the User Experience: Practical Statistics for User Research*. Morgan Kaufmann, March 2016.

[38] Benjamin Schaaf. Sublime Text 4 Build 4142. `https://www.sublimetext.com/blog/articles/sublime-text-4142`, November 2022. Accessed: 2023-02-15.

[39] Egbert Schlungbaum and Thomas Elwert. Automatic User Interface Generation from Declarative Models. In *Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces (CADUI 1996)*, pages 3–18, Namur, Belgium, June 1996. `https://api.semanticscholar.org/CorpusID:10448424`.

[40] Mads Soegaard. The Power of White Space in Design. `https://www.interaction-design.org/literature/article/the-power-of-white-space`, 2021. Accessed: 2023-06-22.

[41] Pedro Szekely, Ping Luo, and Robert Neches. Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 1992)*, pages 507–515, California, USA, June 1992. `https://doi.org/10.1145/142750.142912`.

[42] Matthew James Taylor. Responsive Font Size (Optimal Text at Every Breakpoint). `https://matthewjamestaylor.com/responsive-font-size#:~:text=The%20consensus%20is%20mobile%20font,large%20devices%20is%2018px%20%2D%2020px.`, April 2023. Accessed: 2023-06-22.

[43] Benjamin Ternes, Kristina Rosenthal, and Stefan Strecker. User Interface Design Research for Modeling Tools. *Enterprise Modelling and Information Systems Architectures (EMISAJ)*, 15(4):1—-30, February 2021.

[44] Vi Tran, Manuel Kolp, Jean Vanderdonckt, Ives Wautelet, and Stéphane Faulkner. Agent-Based User Interface Generation from Combined Task, Context and Domain Models. In *Proceedings of the 8th International Workshop on Task Models and Diagrams for User Interface Design (TAMODIA 2009)*, pages 146–161, Berlin, Heidelberg, September 2010. `https://doi.org/10.1007/978-3-642-11797-8_12`.

[45] Vi Tran, Jean Vanderdonckt, Manuel Kolp, and Yves Wautelet. Generating User Interface for Information Applications from Task, Domain and User models with DB-USE. In *Proceedings of the 1st International Workshop on User Interface eXtensible Markup Language (UsiXML 2010)*, pages 184–194, Berlin, Germany, June 2010.

[46] Andries Van Dam. Post-WIMP User Interfaces. *Communications of the ACM*, 40(2):63–67, February 1997. `https://doi.org/10.1145/253671.253708`.

[47] Jean Vanderdonckt. Knowledge-Based Systems for Automated User Interface Generation: the TRIDENT Experience. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 1995)*, volume 95, Colorado, USA, April 1995.

[48] Jean Vanderdonckt and François Bodart. Encapsulating Knowledge For Intelligent Automatic Interaction Objects Selection. In *Proceedings of the INTERACT and Conference on Human Factors in Computing Systems (CHI 1993)*, pages 424–429, Amsterdam, The Netherlands, January 1993. `https://doi.org/10.1145/169059.169340`.

[49] Linn Vizard. What Does A UX/UI Designer Actually Do? `https://xd.adobe.com/ideas/career-tips/16-experts-explain-ux-design-and-what-they-do/index.html`, October 2018. Accessed: 2023-02-15.

[50] Dongyue Wang. Prototype Design of Landslide Forecast System Based on Axure. In *Proceedings of the 3rd International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE 2020)*, pages 1–4, Shenzhen, China, July 2020. `https://ieeexplore.ieee.org/abstract/document/9131237`.

[51] Junfeng Wang, Zhiyu Xu, Xi Wang, and Jingjing Lu. A Comparative Research on Usability and User Experience of User Interface Design Software. *International Journal of Advanced Computer Science and Applications*, 13(8), January 2022. `http://dx.doi.org/10.14569/IJACSA.2022.0130804`.

[52] Brad Vander Zanden and Brad A. Myers. Automatic, Look-and-Feel Independent Dialog Creation for Graphical User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 1990)*, pages 27–34, Washington, USA, March 1990. `https://doi.org/10.1145/97243.97248`.