



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

INTEGRATION OF USER-DEFINED GESTURE INTER- ACTION INTO END-USER AUTHORIZING TOOLS

BRAM DEWIT

Academic year 2022–2023

Promoters: Prof. Dr. Beat Signer
Dr. Audrey Sanctorum

Faculty of Sciences and Bio-Engineering Sciences



VRIJE
UNIVERSITEIT
BRUSSEL



Afstudeer eindwerk ingediend in gedeeltelijke vervulling van de eisen voor het behalen van de graad Master of Science in de Ingenieurswetenschappen: Computerwetenschappen

INTEGRATION OF USER-DEFINED GESTURE INTERACTION INTO END-USER AUTHORIZING TOOLS

BRAM DEWIT

Academiejaar 2022–2023

Promoters: Prof. Dr. Beat Signer
Dr. Audrey Sanctorum

Faculteit Wetenschappen en Bio-ingenieurswetenschappen

Abstract

Gestures are a different way of interacting with an application that does not have to involve the UI of the application. They can be seen as more intuitive in certain use cases, but in current systems, gesture interaction is often not uniform across different applications which can make the gesture interaction less intuitive to the end user. A possible solution to this problem is to allow the user to define their own gesture interaction. The focus of this thesis lies on user-defined gesture interaction, where the user can choose a gesture from a fixed set of gestures (gesture vocabulary) and define which interaction gets triggered by performing the gesture. We also look at user-defined gesture interaction in the context of end-user authoring tools, which are tools that allow the user to author applications, which then subsequently allows for more complex and richer interactions to be defined for each gesture.

In related work, we can find many challenges that one can face when dealing with gesture interaction. For one, the gesture vocabulary and gesture recognition come to mind. Gesture recognition is the recognition of a gesture by the system, and its accuracy is highly dependant on the gestures chosen for the gesture vocabulary. Not only the technical aspects are important, but also the intuitiveness and recall of gestures in users is important, as well as the ergonomics of performing the gesture. Keeping the best practises of gesture interaction in mind is important when designing the interactions, but when an end user is allowed to define the interactions, they might not adhere to these best practises.

The customisation of the gesture interaction is an advantage in some areas, but it also creates some challenges that are important to keep in mind when allowing user-defined gesture interaction. That is why in this thesis, we identify these challenges, not only the existing ones, but more importantly, the challenges that are inherent to the fact that the gesture interaction is user-defined. After identifying the challenges and potential of user-defined gesture interaction, some non-functional requirements are defined for creating a better way of allowing user-defined gesture interaction. We have proposed a set of design principles for designing user-defined gesture interactions, under which each of these requirements is categorised. This is done based on solutions and best practises from different fields of research and some problem-specific solutions are also proposed.

Next, these design principles are used to implement user-defined gesture interaction in an existing end-user authoring tool called eSPACE, where the

usefulness of the proposed solutions are tested. The eSPACE authoring tool is focused on cross-device and Internet of Things interactions, which possibly introduces extra complexity into the process of the design. Before implementing user-defined gesture interaction into eSPACE, a survey was conducted where the preferences of respondents, in terms of the user interface of the implementation, were assessed. The implementation itself was then done according to the preferences of the respondents, in combination with the proposed design principles.

After completing the implementation, an evaluation was done in form of a user study. Participants of the study were given different scenarios in which they were asked to complete certain tasks using the application. The system usability of the implementation was then assessed in a quantitative and qualitative manner, to not only assess the current usability, but also to discover further improvements that could still be made to the system. As it turns out, the proposed design principles helped at giving the design direction and at avoiding some of the earlier identified challenges. The results of the user study were positive about the usability of the design, although a small flaw, and consequently, a possible improvement to the system was found.

What we found from the survey is that in some cases personal preferences of people can vary a lot, due to different reasons, which is why it might be positive for the general user experience to allow the user to customise their experience, through the customisation of gesture interactions, or other interactions in the applications they use. Although the design we made was not perfect, and there will always be a trade off between following best practises and allowing the user to customise their experience, the ideas provided in this thesis can help at creating a more intuitive and easy-to-use gesture interaction.

Acknowledgements

First and foremost, I would like to thank my promotor and supervisor Dr. Audrey Sanctorum for the guidance throughout the process. Throughout the year you have made time to meet and give feedback or guidance and tried to put me on track with ideas of what to write or look up next. Whenever I would ask for pointers on what to do next, you would provide me with ideas to help me continue to move forward. I would also like to thank my other promotor Prof. Dr. Beat Signer for providing feedback at multiple stages throughout the process and providing solutions to guide me to a better thesis.

Next I would like to thank my parents for not only the moral support, but also input, ideas and different perspectives on the work I was doing. They provided me with the courage to move on with this and without them I would not have brought this research to an end.

Last, but not least, I would like to thank my family and friends for the moral support throughout the whole process. Everyone has been supportive and mindful of the process and struggle that comes with writing a thesis and my friends have provided me with much needed distractions at times where it could all feel a bit overwhelming.

Contents

1	Introduction	
1.1	Problem Statement	3
1.2	Research Question & Objectives	3
1.3	Thesis Outline	4
2	Related Work	
3	Design Principles	
3.1	Incentive	14
3.2	Clarity	16
3.3	Intuitiveness	18
3.4	Distinctiveness	19
3.5	Consistency	19
3.6	Interaction Level	21
3.7	Safety	22
4	Survey	
4.1	Setup of the Survey	26
4.1.1	Questions	27
4.2	Results	31
4.2.1	Extra-UI	32
4.2.2	Interaction Rules	34
4.2.3	Gesture Catalogue	35
4.2.4	Editing Gestures	36
4.3	Conclusion	37
5	Implementation	
5.1	eSPACE Authoring Tool	39
5.1.1	App View	41
5.2	Implementing Gesture Interaction	42
5.2.1	Gesture Support	42
5.2.2	Incentive	43

5.2.3	Location of Definition	44
5.2.4	Interaction Levels	45
5.2.5	Clarity	46
5.2.6	Safety	46
5.3	Practical Implementation Details	48
5.3.1	Gesture Recognition Library	48
5.3.2	The Gesture Vocabulary	51
5.3.3	Implementing Gesture Interaction	52
5.3.4	Implementing the Extra-UI	53
5.4	Evaluation	55
5.4.1	Tasks	56
5.4.2	Post-Study System Usability Questionnaire	59
6	Conclusion and Future Work	
6.1	Conclusion	62
6.2	Future Work	64
A	Survey	

1

Introduction

In current systems, gesture interaction is often not uniform across different applications which can be confusing and not very intuitive to the end user. It might not be clear which gestures are possible to use and gestures that are used for a certain purpose in one system might have a different purpose in another. One possible solution would be to allow the user to define what interaction they want to happen for each of the available gestures. A lot of research already exists about gesture interaction and design. There is extensive research on user preferences, gesture recognition, best practises, pitfalls, and technical limitations. This knowledge can be used when designing the implementation of user-defined gesture interaction to guide the user towards a better interaction design, while still allowing the desired level of customisation. Research on custom gestures created by users has already been done, but this is outside the scope of this thesis, where we solely focus on letting the user customise the interactions triggered by a gesture in a predefined set of gestures.

In this thesis we focus on user-defined gesture interaction in end-user authoring tools, which are tools that allow end users to create their own applications. The users of these tools have experience defining interactions in the applications they create and the addition of gestures creates more opportunities for customisation, but certain challenges also arise with it. To

create the best user experience, the authoring tool can implement features to guide the user to a better design of their defined gesture interactions. Before knowing how to guide the user, it is important to figure out which challenges can arise when the user defines their own gesture interaction.

Some of the challenges we will face with user-defined gesture interaction are also challenges of just gesture interaction in general. The disadvantages and challenges of gestural interfaces, when compared to regular user interfaces, have been listed plenty of times by different researchers in different publications, for example an article by Norman and Nielsen [21] calls gesture interfaces “a step backward in usability”. One of the main disadvantages of gestural interfaces is the invisibility of these interfaces. What is meant by “invisibility” is that in regular user interfaces, the components that can be interacted with, which trigger the interactions of the application, are visible on the screen, which means that the user can see which actions are possible with just a glance at the screen. Also, when a user interacts with one of the components of a regular user interface, this component can give visual feedback that it is being interacted with. With gesture interfaces, this is not the case. The possible gesture interactions do not have to be visible on screen, which can be argued to be an advantage in terms of on-screen clutter, but in terms of usability, the regular user interface will be easier to understand and use. Especially because the gesture interactions will have to be remembered by the user if there is no visual indication to identify them by. There are solutions to these problems to improve the visibility of these gesture interfaces, but in terms of visibility, the regular user interfaces will have an advantage over gestural ones.

Even though there might be innate disadvantages to (user-defined) gesture interaction, there are also advantages and these advantages could outweigh the disadvantages, especially for certain specific use-cases. The incentive for implementing a gestural interface is therefore important and should be considered on a case-by-case basis to see if the advantages outweigh the disadvantages. With the identification of the challenges and the proposition of solutions to these challenges, we aim to mitigate these disadvantages as much as possible.

1.1 Problem Statement

The advantage of letting users define gesture interactions is that they can choose how they will interact with the application they create, which allows them to define the gesture interaction to be more intuitive to themselves. A more intuitive gesture interaction allows for easier recall of the interactions which can result in a speedup of the workflow of the user and a lower rate of error in the usage of the gestures. One problem that arises is that the user might not have any knowledge about the topic of gesture interaction, and some of its pitfalls, thus it might not be trivial to create an easy-to-use and intuitive way of defining gesture interactions for these users. That is why the application should try to help them create good interactions and help them avoid these pitfalls.

We will explore the challenges and possible solutions of not only gesture interaction, but specifically user-defined gesture interaction. Not only do gesture interactions bring challenges with them, the fact that they are user-defined poses questions in terms of how the interactions will be defined, where in the application they will be defined, how to prevent clashes between certain interactions with the possible large number of combinations between the gestures and the effects they can trigger. Although plenty of research exists about gesture interactions, the user-defined aspect we will discuss, has not been thoroughly covered, such that there are no guidelines on what could go wrong when implementing user-defined gesture interaction, what to look out for, and how to create an interface for the definition of these interactions which is intuitive and easy-to-use. Also is there a discussion to be had about when the implementation of user-defined gesture interaction should even be considered in an application.

1.2 Research Question & Objectives

This thesis tries to provide an answer to the question “How to implement user-defined gesture interaction in an easy-to-use and intuitive manner?” Answering this question is approached by identifying the challenges that arise when the end user is allowed to define the gesture interaction of their applications, and by providing ideas to solve these challenges.

Besides identifying challenges and proposing solutions to these challenges, we also want to propose a set of non-functional requirements to which implementations of user-defined gesture interaction have to satisfy in order to be

as intuitive and easy-to-use as possible. These requirements will be posed in the form of design principles to consider when designing an implementation of user-defined gesture interaction. Besides providing theoretical principles which could improve the implementation, we also want to use these principles in a practical implementation, where possible flaws or areas of improvement can be discovered, either during the implementation, or when evaluating the implementation afterwards on the basis of a user study.

1.3 Thesis Outline

In this thesis, first, related work on the topic of gesture interactions and user interfaces is explored. As we already established, many challenges that occur when implementing gesture interaction, will also occur when implementing user-defined gesture interaction, which means that familiarising ourselves with existing research on this topic will help us in finding all challenges that may arise, and in coming up with solutions for these challenges. Because of the user-defined aspect of our gesture interactions, different challenges may be posed when trying to implement an intuitive and easy-to-use way of defining and using these interactions. Therefore, we will identify the possible challenges when implementing user-defined gesture interaction next, after which, for each of these challenges a set of solutions get proposed.

Because the optimal solution to certain challenges is different depending on the use case of the authoring tool, these challenges and solutions are then used to distil a set of design principles, which should help at creating a more intuitive and easy-to-use way of implementing user-defined gesture interaction. Lastly, these ideas are applied to a specific use case, namely the eSPACE authoring tool [26], to demonstrate these principles in a real life scenario and to evaluate their usefulness.

Before starting the implementation, a survey was done where participants were asked about their preferences in terms of some of the user interface design decisions, to assess what they think is intuitive in terms of an interface used to not only define user-defined gesture interaction, but also to visualise the gestures that can be used in the interactions, and to help remember the interactions that were defined. After interpreting the results of the survey, the implementation was done according to the results of the survey, but also according to the earlier defined design principles. Each of the principles is discussed in the context of our implementation, after which the more technical details of the implementation are discussed. After the implementation

was done, a user study was conducted to get quantitative and qualitative feedback on the usability of the system. The results of this user study were then used to evaluate the system that was implemented and to come up with possible improvements that could be made.

Lastly, the conclusion summarises everything that was discovered during this study, and answers to the posed research questions are provided, after which, future work that could further build upon and improve our research is discussed.

2

Related Work

When it comes to gestures, there are important distinctions to make and differences to recognise. A gesture is a movement that expresses some meaning or intent and in the context of (smart) devices, this gesture is an interaction with such device. This gesture first has to be recorded and second it has to be recognised by the system to know which gesture the user has performed. Because of the many possible types of gestures, there are also many ways to record and recognise these different types of gestures.

In general we can divide gestures into two broad groups, namely the so-called “touch” gestures and “mid-air” gestures. Touch gestures are, like the name suggests, gestures where the user touches a surface of the device, either with their hand or a specialised tool like a pen, which records the movements of the user’s finger or the tool. If multiple points of the surface are touched at the same time while performing a gesture, these touch gestures are also called “multi-touch” gestures. Mid-air gestures are gestures where the gesture is not expressed by touching the device, but rather by making a mid-air movement with one’s limbs, usually hands, which can be recorded using various different sensors. Both mid-air and touch gestures have their advantages and drawbacks, and it is up to the designer of the gesture interaction to decide which type of gestures are more intuitive in their specific use case [7].

Often times, devices that use multi-touch gestures use a touch screen to recognise the gestures, which has the advantage that the user can directly touch the elements on the screen when they want to interact with them, which makes the interaction more intuitive to the user. Another advantage is that a touch screen replaces the need for several other buttons or another touch surface on the device, which saves space. Touch screens are found on devices such as smartphones. A disadvantage of using gestures is that it is not easy to know which gesture can be performed as they are not shown in the interface of the application, unlike buttons for example. The applications for gestures we are focusing on in this research are applications with screen interfaces, this does not mean the gestures of the end-user authoring tool are only touch interactions via a touch screen, because recognising mid-air gestures is still possible through other sensors in the devices, but it does mean that virtual reality (VR) or Augmented Reality (AR) gesture interactions [14, 9] are outside of the scope of this research.

With (multi-)touch gestures, the gesture starts when the surface is touched and ends when the user stops touching the surface. This means the start and end of the gesture are quite well-defined, which helps with the recognition of the gestures. The recognition of gestures is usually not very straightforward, this is something that a 1999 study already drew attention to [15] and even though gesture recognition techniques have evolved since then, mainly thanks to machine learning technology, the recognition of gestures still is not trivial. Many gesture recognition algorithms and frameworks have been developed over the years and advancements in accuracy and versatility have been made. For (multi-)touch gestures, a recent survey by Magrofuoco et al. [18] discusses the state-of-the-art recognition algorithms. The paper identifies 16 different relevant recognisers and discusses the algorithm, performance and properties for each of them. It points out that many recognisers classify gestures by using machine learning techniques, which makes sense as they are highly accurate and robust to noise, but they require training with a significant number of examples and domain knowledge, which is not always a given in situations like end-user gesture customisation and rapid prototyping of gesture based UIs as the paper points out. The disadvantages of machine learning approaches have also been researched and pointed out by others [11, 24]. For these reasons, algorithms other than machine learning still exist as gesture recognisers, which the paper by Magrofuoco et al. [18] divides into three categories (feature-based, template based, and metrics based recognisers), each with their own advantages and disadvantages.

When it comes to mid-air gestures, there is more variation in the different types of gestures and tools when compared to touch gestures. A mid-air gesture can for example be an arm movement, which can be recorded in various different ways using accelerometers in an armband, a glove, a handheld controller, a smartphone, etc. An arm movement can also be recorded through a webcam, or even via electromyography, which is a technique used to measure electrical activity in the muscles [25]. Besides arm movements, there is also the option of recognising symbolic hand gestures like sign language, or even full-body movements. The hand gestures can be recorded by a glove and the full-body movements could be recorded via a body suit, but more convenient is also if the movements are recognised by a camera. Webcams are common and relatively cheap hardware with which many types of gestures can be recognised, but it is hard to perceive depth using a normal camera. To solve this problem, technology like the Microsoft Kinect can be used, which is a device which combines an infrared laser projector with an infrared camera and an RGB camera. The infrared laser and camera are then used to measure the depth of the image which allows for accurate gesture recognition [6]. Multiple other technologies exist for mid-air gesture detection, like for example the Soli project by Google [13], which uses a millimeter wave radar to detect gestures, but going over all of these technologies lies outside the scope of this thesis.

When considering gesture recognition, another challenge arises for mid-air gestural movements. In contrast to touch gestures, where a gesture starts when the surface is touched and ends when the surface is not touched anymore, it is harder to know when a mid-air gesture starts and ends. It is important to know when a gesture starts and ends to know whether or not a certain movement was an intentional gesture or not. For example when mid-air hand movements are being recognised, we do not want to classify a transitional hand movement as an intentional gesture. This can easily be avoided by letting the user press a button for the duration of the gesture, but that approach also limits the potential ways the user interacts with the system. In certain situations, a more preferable approach would be that the user's movements are continuously being monitored, but that the system can recognise an intentional gesture from an unintentional gesture. Those kinds of systems exist [2] and new ones are continuously being made and improved upon.

The quality of gesture recognition depends on the performance of the gesture recognition algorithm, but this is not the only factor that affects

recognition. Another major factor in the recognition of gestures is the defined gesture vocabulary. The gesture vocabulary is the set of gestures that can be used. Even if a certain algorithm can recognise the use of a single gesture with a very high accuracy, if the gesture vocabulary contains another gesture which is really similar to that first gesture, the algorithm might confuse one gesture for the other, which would be very bad for the recognition accuracy. Because of this phenomenon, differentiation between gestures in the gesture vocabulary is important, but not only is it important for the recognition of gestures by the system, it also helps the user in differentiating between different gestures. Other human aspects like intuitiveness of gestures and fatigue play an important role in the design of gesture vocabularies, and extensive research has been done on these topics [31].

Another possibility that is outside the scope of this thesis, but which is still relevant as a possible extension to this work, is the possibility of adding custom, user-defined gestures to the application. This end-user gesture customisation would add more possibilities to the gestures that can be used, but it would introduce many challenges as well [23]. These custom gestures could be created and defined using different paradigms, like for example through demonstration [16, 17], or even options like defining multi-touch gestures as regular expressions are explored [10].

Knowing how gestures can be defined and what the challenges and potential are of custom gestures is important when creating a paradigm for user-defined gesture interaction in end-user authoring tools, because this paradigm will have to be compatible with different kinds of gestures and possible extensions like the custom user-created gestures. Knowledge of gesture recognition, and its possible pitfalls, is important when selecting the set of gestures that the user will be able to use for their custom gesture interactions. The selected set of gestures has to be distinct enough to humans, but also to the computer, such that a high accuracy and low false-positive rate can be achieved when using the gestures in the application.

Although plenty of research on gestures created by users exists, most of that research focuses on the creation and recognition of the gestures, which is an aspect that introduces many challenges that for this research, we are not interested in. That's why in this thesis we focus on user-defined gesture interaction with a predefined set of gestures, more specifically for end-user authoring tools. This allow us to look at the interaction between the user-defined gestures and the user-created applications, and lets us focus on

allowing and guiding the user to create good gesture interactions for the applications they create.

3

Design Principles

In this chapter, design principles for user-defined gesture interaction are proposed as a set of non-functional requirements used in the implementation of the user-defined gesture interaction. This is done by identifying challenges that can occur when implementing the user-defined gesture interaction into an application and by providing possible solutions to these challenges based on existing concepts from the field of gesture interaction, but also concepts from different fields of research where solutions to similar challenges were created. Each of the broader principles and topics are discussed in a separate section where we will look at how allowing user-defined gesture interaction affects these principles. Allowing the end user to define the gesture interaction of their application has advantages, but we will also see that some of the conventional principles for gesture interaction are harder to enforce once the user is allowed to define their own interaction. Each of the discussed principles and topics is summarised as a keyword that is used as the title of the section, which makes it easier to refer to it later.

The focus will lie on customising the interactions of a predefined set of gestures, meaning that the designer of the end-user authoring tool will provide the user with a set of gestures, which the user can then use to customise which interactions will happen, depending on which gesture was performed. This means the use of custom user-defined gestures is out of the scope of

this thesis, because allowing the user to use custom gestures they invented brings a whole new set of challenges with it, as it makes it hard to enforce the best-practices and principles of gesture interaction design. In this chapter we will discuss which properties a good gesture vocabulary has and why these properties are important, after which it will be clear that these properties are hard to enforce when the user can create their own gestures.

3.1 Incentive

The very first thing someone should think about while developing the gesture interaction for a certain application is the motivation behind why gestures should be used for these interactions altogether. The usage of gestures should be motivated on why it would be better than for example just using a button in the interface. There are multiple possible reasons why gestures would be a better approach than using a user interface (UI) element. An example of a reason why touch gestures would be better than using an element in the UI is the fact that a gesture does not take up space in the UI. This could leave more space for other UI elements, which in some cases is very desirable. Another reason could be because a gesture could be faster to perform than clicking on a UI element, for example, when this UI element takes multiple clicks to activate, like when it is situated in another menu. The application could in the case of this example even give the user both options, such that if they prefer the faster gestural approach they can use it, or otherwise they can use the slower, more conventional UI interactions.

This idea of having simple and obvious methods of interaction, together with less obvious, but faster methods of interaction for more expert users, is a familiar concept in user interfaces and can definitely be applicable when designing gesture interaction, but the designer should facilitate the user's transition to the more efficient interaction methods [4, 12]. One could also argue that if the UI is less cluttered because some interactions are performed by gestures, it will be easier to find a certain button in a UI because there is less visual clutter. While this might be true for some cases, gesture interaction has a learning curve, so instead of learning the gestures, a user could learn the UI layout, which could nullify the need to use gestures for this reason, but the gestures could be easier to learn to some users, and of course the other advantages still apply.

For mid-air gestures, some of the same principles apply, like using gestures to save space in the UI or having faster interactions than when using the UI, but these can be taken to another level as now the user might not need to touch or get close to the device at all. Like already discussed in Chapter 2, there are many different technologies for mid-air gesture recognition, either with some sort of tool that needs to be carried like a controller or armband, or with a device that captures the user's movements from a distance like a camera or radar sensor. Depending on the application, a certain amount of interactions, if not all, could now be done without even touching the device, which can have multiple advantages. In terms of speed, it could again speed up the interactions in the application workflow, for example due to the physical freedom of the device not needing to be touched, something that has for example been a concept used in smart home technology [8].

The intuitiveness of using gestures over standard UI interactions is something that could be an incentive for using either touch or mid-air gestures. For touch gestures, it could be more intuitive for a user to use a swipe gesture to navigate between different virtual desktops on a laptop instead of remembering and pressing an arbitrary keyboard shortcut. For mid-air gestures, it is even more clear that they could be a more intuitive replacement of, or supplement to, the conventional methods of interaction. When using gestures in real life for conversation, these are mid-air gestures, using mainly the person's hands to indicate a certain meaning or message. This fact can be used in the design of gesture interaction, as mid-air gestures might be more natural to the user than other forms of interaction, but the designer still has to make sure that the interactions are intuitive and add value to the application.

Lastly, this thesis is not just focused on implementing gesture interaction, but implementing user-defined gesture interaction, which means not only the reason for using gestures in an application, but also the reason for making this customisable to the user, has to be considered. There are obvious advantages to user-defined gesture interaction because if users can define their own gesture interaction, it is made to their preferences, which will make it more intuitive, easy to use and easier to remember to the user. In end-user authoring tools, the goal might be to give the user as much possibilities as possible, as they are the designers of their own creations and they should be able to decide to use this feature or not. Even if this is the case, there are disadvantages to this extra level of customisation. One example could be the well-known "Paradox of Choice" [27] which introduces the idea that more choice is not always better because it increases the effort needed to make a decision which can lead a person to make the wrong choice. Like we

have discussed, and will discuss more in this thesis, the design of gestural interaction has certain best-practises, principles and pitfalls that should be considered, but users with little to no domain knowledge might fall victim to certain pitfalls and fallacies due to the fact that they have the option to customise these interactions. While later in this chapter we discuss some ways in which the designer of an authoring tool can aid users in creating better gestural interactions, it should still be considered if the advantages of user-defined gesture interaction outweigh the possible disadvantages.

3.2 Clarity

Earlier we already slightly touched upon the point that gesture interaction is not always visible at first sight, especially when one of the advantages of using gestures is that it can reduce the visual clutter in the UI. When designing gesture interaction and especially with user-defined gesture interaction, it is important to think about how to make clear to the user which gesture does what and which possible gesture interactions there are. Another part of clarity is that it should be noticeable to the user when a certain interaction gets triggered, as this is not always evident depending on which kind of interaction got triggered.

There are multiple solutions to solve or improve the first challenge of making clear what the possible gesture interactions of the application are. One that stands out are so-called “Extra-User Interfaces” (Extra-UIs). Extra-User Interfaces (Extra-UIs) are separate user interfaces giving extra information about the use of the current application interface. These Extra-UIs could be used to show the current defined gesture interaction of the user’s created application, and it would even be possible to let the user use this extra interface to add, change or delete gesture interactions. When creating such an Extra-UI, it is important to know some of the best-practises and user preferences for Extra-UIs to be able to create an easy-to-use and intuitive user experience. The topic of user preferences for Extra-UIs is studied and evaluated in Melchior et al. [19].

Another use for Extra-UIs in the context of gesture interaction is the demonstration of the defined gesture interaction through visualisations in the interface. If a certain element of the application has certain custom gesture interactions, it might not immediately be completely clear which gestures will trigger a certain interaction and what that interaction would be. There are multiple ways this could be communicated to the user, for

example through text or icons in the Extra-UI, but another possibility is to create visualisations that show the interactions by demonstration, a solution that is explored in Vanacken et al. [28] and Walter et al. [29].

Showing the effect by demonstration might be hard to accomplish for applications created via an end-user authoring tool because of the freedom a user gets in creating gesture interactions. Explaining the interaction with text or even icons seems like a more feasible approach. It would also be a possibility to just show the gesture interactions in a list to the user, when this user requests it, where the gestures and interactions have a name that is written in text or is portrayed with an icon. Another consideration would be how and when this Extra-UI is shown on screen. The most obvious approach is to place a (small) button on the UI that shows the list when clicked, this is a simple solution but it could work quite well. Another option is to let the user use a gesture to reveal the Extra-UI, but this might not be a good solution, as the Extra-UI is made to help people remember the gesture interaction, but then an invisible interaction would have to be used to get to the UI that shows which invisible interactions there are in the application. A last approach is to show the interactions at certain points in the application workflow where users might need the gesture, but this is again something that is hard to impose on end-users who create their own applications and even if they do use this feature, best-practises and design principles might not be used.

One thing that also affects the importance of clarity when it comes to gesture interaction in applications created using an end-user authoring tool, is the target user of that application. If the tool is used by users to create applications that will be used by other people, it might be in the interest of the user if the tool allows them to type out a good description of the interaction that triggers when each gesture happens. They themselves might know which gesture does what, but the people using their applications might not understand exactly what a gesture does, just from a few or a single keyword(s). That is why in this case, it would be beneficial if the tool allows the option of having a description for each interaction, but not only allowing it, also suggesting or even obligating the user to do so.

In contrast, when the end-user authoring tool is focused on users who will create applications for their own use, it would still be a useful feature to allow the user to give a description of the interaction, but it becomes less important than when the application is also used by others. In this case,

the user made the application, which means that they are more likely to remember what a certain interaction does by just one or a few keywords. Like already mentioned, it would still be desirable to give the user the option to give a good description, as they could just completely forget what a certain interaction does, or they could still share the applications with others. Like for example when there is an end-user authoring tool that allows users to create applications for their own smart home system, they would maybe want to share the applications with the other people in their household. Assumptions like this about the use of the tool should be avoided, even if the large majority of users will use it in a certain way, there will always be people who would suffer the consequences of our assumptions [5].

3.3 Intuitiveness

What is intuitive for one person, is not always intuitive to another. Factors like culture, technological literacy and familiarity with certain technologies can all play a role in how intuitive a certain gesture is to a person. Technological literacy is usually lower with older adults and several studies have been done to find out how gesture interaction can be made more intuitive for them [3, 20]. Gestures could be more intuitive if they are metaphorically and iconically logical towards functionality, but due to different cultures, languages and their natural gestures, it is a challenge to come up with gestures that seem intuitive to everyone. Although this is true, gesture interfaces already exist in many applications and fields, which means the designer of the gesture interaction can follow common patterns that will be recognisable to users who are already familiar with certain gesture interfaces. Even when common patterns exist, there will still be users who are not familiar with these patterns, who could benefit a lot from good clarity on the possible gesture interactions, like discussed in the previous section.

All of this is an argument in favour of user-defined gesture interaction, if the user is the only person who will use this gesture interaction, because the interaction they defined will feel intuitive to them. If some else than the user who designed the gesture interaction can also use the application, the gesture interaction might not be intuitive to this other user. This is mainly because it is hard to enforce that the user defines gesture interaction which uses metaphorically and iconically logical gestures for the defined interactions, and of course also because like we discussed, different things are intuitive to different people. Therefore, this again shows the importance of providing the user with a good set of gestures that are easy to perform and

remember. There is plenty of research on the topic and guidelines have been made for developers of gesture vocabularies. Not only user preferences and best-practices for intuitive gestures have been researched [1], but also the ergonomics of gestures, to find out which gestures are easy to perform, and perform for an extended amount of time or a large number of times.

3.4 Distinctiveness

Another important property to keep in mind when designing a gesture vocabulary is the distinctiveness of the gestures between themselves. The main reason why the gestures of the provided gesture vocabulary should be distinct enough is for gesture recognition by the system. It's easier for a gesture recognition algorithm to differentiate between two gestures if they are not too similar to each other. Similar gestures lower the accuracy of the recognition algorithm, but there are also other factors in play when it comes to the accuracy of gesture recognition. Another important factor is the amount of gestures in the gesture vocabulary. If the amount of different gestures in the gesture vocabulary gets larger, at some point the accuracy of recognition will start to suffer, but if the gestures are distinct enough, the system will be able to reach the same accuracy at a larger amount of gestures. This allows the designer of the authoring tool to provide the user with more gestures at their disposal.

Having a gesture set with distinct enough gestures is not only important for gesture recognition, but it might also help the user remember what each gesture does [1]. Additionally, if two gestures are too similar, the chance of the user performing the wrong gesture is also greater than if they would be more distinct. Some other factors like familiarity with certain gestures might play a bigger role than distinctiveness of gestures when it comes to the user's recall of the gestures and their perceived intuitiveness of the gestural interactions, but it is still a factor that could be kept in mind to improve the gesture vocabulary.

3.5 Consistency

When talking about consistency, we not only want to point to consistency in the usage of gestures, but also consistency in the end-user authoring tool when defining the gesture interactions. Consistency in the usage of gestures is hard to enforce when a user can define their own gesture interaction. One

possible way in which the tool could aid the user in creating more consistent gesture interactions is by providing suggestions, similar to the idea used in the DesignScape tool by O'Donovan et al. [22]. When the user would create a gestural interaction, the tool could either suggest other interactions, or if multiple interactions are made, it could suggest similar gestures for similar purposes. This all seems like a great way to help the user create the interactions, but practically it is not straightforward to implement such a suggestion system, especially if the tool provides a wide set of possible gestures, maybe even a combination of mid-air and touch gestures on possibly a combination of different devices.

This again circles back to the point of *incentive* that was discussed in the first section of this chapter. The consistency of gesture interactions might be desirable in applications when considering user-friendliness, but in the case of these end-user authoring tools, removing the option of user-defined gestures will consequently remove certain gesture interactions altogether. Considering this, if the designer of the tool has enough incentive to let the user define gesture interactions, the consistency of these interactions might be an afterthought if it would otherwise mean there would be no custom gesture interactions at all. However, even though it may be an afterthought in some situations, it is still a factor to keep in mind and possibly improve on when allowing user-defined gesture interaction.

The other point of consistency to consider is when it comes to how the user can define the gesture interactions in the tool. The definition of gesture interaction should be well-integrated into the tool, which means the designer of the tool should consider not breaking the patterns that the tool has provided the user with when designing other kinds of interactions. If the definition of gesture interaction works the same as, or similar to the other forms of interaction design that already exist in the tool, it might be easier for the user to learn how to design gesture interaction because of the similarity to the other design methods they are already familiar with. Not only can the learning process be sped up, but depending on how the tool is designed, defining gesture interaction in the same interface as other interactions might speed up the workflow as opposed to defining the gesture interaction in another interface. Sometimes this is not possible, or maybe not even desirable because of the design philosophy of the tool. It stays the responsibility of the designer of the tool to preserve consistency, which sometimes means not making things look similar because they are inherently dissimilar things. Lastly, there is also the consideration between preserving consistency and speeding up the

workflow by allowing the definition of gestures in places where, according to some, they should not be allowed.

3.6 Interaction Level

Another important consideration to make when allowing user-defined gesture interaction is on which level of the application the interactions will be defined. Gesture interaction with an application can happen on multiple levels of the application. For example, the user could define gesture interactions for their application where they interact with certain UI components using gestures, like for example “long pressing” a UI element to trigger a certain interaction. In contrast, the user could also define gesture interactions that can be used throughout the whole application and trigger their action whenever or wherever they are used. Another option is that gesture interaction could be defined for a single view in the application, if the authoring tool supports creating applications with multiple views, such that the gesture triggers that interaction when used in this particular view and not in another one. When allowing users to define gesture interaction, the decision has to be made on which of these levels the interactions will take place. Will the gestures be used throughout the whole application, only in a certain view, only on (certain) UI elements, on multiple of these levels, or all of them?

This decision lies with the developer of the authoring tool and has an implication on multiple factors which are discussed in this chapter and different decisions could bring different challenges with it. First of all, depending on which level the interactions happen, the place where to define the gesture interaction might be different. This would be to preserve factors like intuitiveness or consistency in the interaction design. Especially when multiple interaction levels are possible, it might be hard to design the definition of gesture interactions in a way that is consistent to the other paradigms and patterns used in the tool. Secondly, a challenge that may come with allowing interaction in multiple levels of the application, is that clashes or overlap in gesture interaction could occur. These clashes can be prevented, but it can introduce complexity when trying to do so. Overlapping gesture interaction could be prevented easily by letting a user only use a certain gesture once throughout the whole application, no matter on which level it is used. The thing is that, for example, if a user wants to assign a certain gesture to a UI element when this gesture is already being used on an application or view-level, it might not immediately be visible to the user where the gesture is already in use. If all gesture interactions are defined in the same place, this

will not be a problem, but if UI or view-level gestures are defined somewhere else than application-level gestures, it might take the user time to find where the clash happens. This could be solved by, like mentioned before, defining all gesture interaction in the same place, but if that is not desirable, the tool could let the user know where the gesture is being used.

If it is known which device(s) the applications will be used on, it might also be good to know what kind of gestures are already used in the device's gesture interaction to adapt the gesture vocabulary and prevent clashes with these device-level interactions. Lastly, it might just be more complex and less intuitive to a user if different levels of interactions are being used, especially with view-level interaction, a user may not know that in a different view, different gestures can be used. It might be inefficient and lead to frustrations if in one view of the application certain gestures can be used, but not in another one. The trade-off between providing the user with more possibilities, but also guiding them to create better applications is one that reoccurs in many of the topics discussed in this chapter.

3.7 Safety

In a programming context, safety could point to various different aspects of the program, but what is meant here is if a gesture interaction gets triggered by mistake, either by the user or the gesture recognition algorithm, the effect of the interaction is non-critical and reversible. The same principle is applicable to buttons in an interface, a button can be accidentally clicked, which is not a problem for non-critical, reversible actions, but if an action is not (easily) reversible, this means that an accidental click can cause major problems. This is why usually in an interface, when the user clicks a button which will cause some action that is critical in the sense that it is hard or impossible to reverse, a dialog window will pop up that asks the user for confirmation. This does of course not solve the problem completely, because even then mistakes could still happen, but it greatly reduces the chance of it happening. This solution and others could also be implemented when designing gesture interaction to improve the safety of the intentional and unintentional interactions.

One could argue that it is even more important to implement this kind of safety measures with gestures than it is with buttons, because with gestures there is no visible clue of what gestures there are and if they are being performed, accidentally or not. With a button in the interface, there is at least some visual clue of the button being there and the accidental click happening,

even if this can also go unnoticed by a user. Lastly, the gesture recognition could also fail at recognising the right gesture. It might recognise a gesture when no gesture has been performed (false positive), or it might inaccurately recognise one gesture as another. This could be even more dangerous to the user as they might be aware of an accidental gesture they performed, but not of a gesture they did not perform, but is recognised by the system as one. This is why it is very important to create a good gesture vocabulary and recognition with gestures that will most likely not trigger by accident.

Just like before, we are not just focused on gesture interaction in general, but more specifically on user-defined gesture interaction. With user-defined gesture interaction we could force the user to create a message that pops up whenever a gesture is performed, but this would not be desirable in some cases. For example if we want some gestures to be performed quickly, like when they are used a lot and do not trigger a critical action, an extra pop-up asking for confirmation would slow down the usage of these gestures. However, more importantly, for mid-air gestures, sometimes one of the greatest advantages is not needing to touch the device, which would be nullified by having to confirm the action on the device. This confirmation could still be done by another gesture, but the risk of accidentally performing the action still exists, even though it will be significantly lower. If the mid-air gesture is performed using a controller with buttons, these buttons could be used to confirm the action as well, but as we know, there are many ways of recognising gestures without any other form of button controls. This means that for gestures, but especially mid-air gestures, it might not be desirable to have any critical actions as an effect triggered by a gesture, but if we do, the user should have the option to create a confirmation window to prevent accidental triggering of the interactions, because of a mistake made by either the user or gesture recognition algorithm.

4

Survey

For the implementation of user-defined gesture interaction in the eSPACE authoring tool, which is the real-world use case we have done our implementation for, we have not only implemented user-defined gesture interaction in the applications, but we have also made an Extra-UI to help users remember the defined interactions. In this Extra-UI, we do not only let the user view the defined interactions, but we have also designed a UI to let them edit, add and remove these interactions. Lastly, we also have implemented what we call a “gesture catalogue”, which is a screen where all the possible gestures are listed, together with their corresponding names which we use throughout the application. With this gesture catalogue, the user can check which gestures are available, but also what a certain gesture would look like, such that they know how to perform that gesture. To make sure that our design is intuitive to potential users and according to how they would envision using the Extra-UI, we conducted a survey to assess the preferences of potential users when it comes to multiple design questions related to the UI. The survey does not only cover the aspect of user-defined gesture interaction, but any kind of interactions that are defined for the application, as these other interactions could also be integrated into the Extra-UI, such that users can more easily remember the interactions, and possibly edit them as well.

In the first section of this chapter, the setup of the survey will be discussed, which includes giving a description of all the questions that are asked in the survey, together with their possible answers. After that, we take a look at the results of the survey, both the statistical evaluation of the responses, as well as any qualitative evaluation of any written answers. Lastly, the conclusions we can formulate using these results will be discussed and evaluated in the context of our implementation.

4.1 Setup of the Survey

In the survey we first ask some questions about the demographics of the respondents and also about their experience with computer science related courses during their studies. After responding to these questions, the respondent will see a screen with an explanation about the context of our research, explaining its purpose and some concepts that will be covered in the survey. Next, a screen is shown explaining how behaviour is defined for applications created using eSPACE, namely by defining interaction rules specifying which event triggers a certain interaction. Next, an example of these rules is given, together with a mock-up of a possible Extra-UI showing these rules. After this, the main part of the survey starts, where questions about the Extra-UI of our implementation are posed.

Interaction Rules		Edit Rules				
IF gesture:	Triangle	on:	Page	THEN	Light On	✘
IF gesture:	Circle	on:	Page	THEN	Light Off	✘

Figure 4.1: A mock-up of what the Extra-UI could look like with an example configuration where a *Triangle* and *Circle* gesture turn a light on and off respectively

The survey starts with questions about what contents the respondents would like to see in this Extra-UI and about their preferences in terms of certain design decisions. Depending on the answers of the respondents, the next questions and possible responses to these questions will be slightly different to elaborate on their earlier responses. The survey asks a maximum of 13 questions to the respondents ranging from the demographics questions, to questions about the contents of the UI, and questions about the look and feel of the gesture definitions. Some questions and explanations are also clarified using images containing mock-ups of the different parts of the UI and their variations.

4.1.1 Questions

Below we will discuss every question that a user can encounter in our survey, together with the possible answers to these questions. The full survey can be found in Appendix A

Demographic Questions

Like mentioned before, first we asked some demographic questions to get some more information about the target audience of the survey. All responses to the survey are anonymous, but knowing the age group and technological background of the respondents can possibly provide interesting insights into the data.

First we ask the year of birth and the gender of the respondent. After that, we ask what the highest level of education is that the respondent has completed. If the respondent has at least earned a bachelor's degree, they will be asked if they got in contact with programming and/or user interface design during their courses. If the respondent answers positively to the previous question, they will see a question asking how many computer science-related activities they followed. The possible responses are "Not much (1–2 courses)", "A few (3–4 courses)", "Many (I am a computer scientist)".

After responding to this question, there is only one question left in this part of the survey, namely in which country the respondent was born. Like mentioned before, this was the last question of the demographic part of the survey, after which a page with more information about the research is shown, as an introduction to the more substantive part of the survey.

Extra-UI

Like mentioned before, the next part of the survey contains questions regarding the Extra-UI in which the user can look at and configure the gesture interactions of the application. The first question asks the respondent if they would rather have a single screen in the Extra-UI for editing interaction rules, where the defined rules can consequently also be looked at, or if they would prefer a separate rule editing and rule viewing screen. The idea behind this question is that second option would allow the rule viewing screen to have a different layout than the rule editing screen, but from the responses we can find out if the respondents think they would ever need a separate screen for this.

The next question is only asked if the respondent selected the “separate rule viewing and rule editing option”. It asks whether or not the user would prefer the rule editing and rule viewing screen in different tabs of a single pop-up window, or if they would prefer two different screens, one for editing and one for viewing the interaction rules. Some respondents might prefer two separate buttons that open different pop-up windows because the two screens would be used in different contexts and they prefer a separation for this reason, while the single pop-up has the advantage of taking in less space in the UI, because there is only 1 button that is needed. The other advantage of a single pop-up would be that it is easier to switch between the two views, if the user would ever want to do so.

Interaction Rules

Next, we explain how the interaction rules defined in eSPACE follow a so-called *trigger-action* pattern, which means a certain event (like a button click, or a gesture that is performed) triggers a certain action by the application. We also explain how currently rules are categorised by their triggers and how *user-initiated* triggers (like a button click for example) and *contextual* triggers (like a time or location) are defined. The user-initiated interaction type could be divided further into categories like for example *UI interaction*, *gesture interaction*, and *device interaction*. The reason why we explain this concept and the different types of rules is because, to show a more structured overview of the list of rules, they could be grouped into different categories. Because the interaction rules are trigger-action rules, the respondent is asked if they would prefer the rules to be grouped by the type of trigger or the type of action of the rule. This division would be done in the rule viewing or rule editing screen, depending on whether or not there would be a separate rule

viewing screen. Figure 4.2 shows an example of how the interaction rules could be divided by the different types of triggers mentioned earlier, which is also used in the survey to demonstrate the idea of grouping the rules by a common factor like the type of trigger. The question also has an option where the respondent can respond that they don't want the rules to be grouped by type and they just prefer one list of all the rules, and also there is an option for the respondent to select "Other:", which allows them to type another suggestion if they would have one.

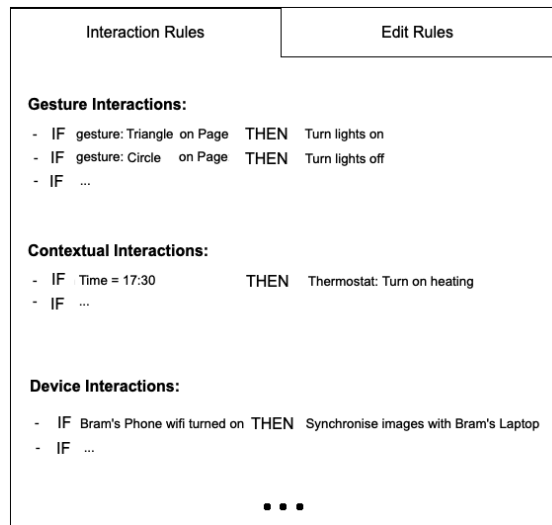


Figure 4.2: A mock-up how the defined interaction rules could be grouped by the type of trigger they are defined for

The interaction rules can be visualised in different ways. In the eSPACE authoring tool there are two different views, used to edit the interaction rules, namely the *Rules* view and *Interaction* view. The first one uses a textual representation of the rules, where the rules can be edited using drop-down menus, while in the interaction view, the rules are displayed in a graphical way and they can be edited using a combination of drag-and-drop functionality, together with pop-up windows where these interactions can be configured further. For our Extra-UI we would go with a textual representation of the rules and because of this, it might make sense to stay consistent with respect to the notation already used in the *Rules* view. This notation uses “IF *trigger* THEN *action*” as a format to display the rules. The survey explains that this is the notation that is currently used and asks if the respondent would prefer the “IF *trigger* THEN *action*” notation, or if they would prefer another notation like for example a slightly more visual notation with an

arrow like “trigger → action”. This question also has a third “Other” option with a text box where the respondent can fill in their own suggestion.

Gesture Catalogue

The previous questions focused on the interaction rules in general and how they would be displayed, but because this research is about user-defined gesture interaction, the next questions are focused on the parts of the Extra-UI that are specifically created for the gesture interactions. The first idea related to gestures, is to add a *gesture catalogue* showing the name and a visualisation of each of the gestures that can be used in the application. This is a solution to the possible issue where the user might not know all possible gestures that can be performed, or what they would look like. This idea is also explained in the survey. In Figure 4.3 a mock-up of what such a gesture catalogue could look like can be seen. This mock-up is also shown to the respondents of the survey, but based on their previous responses, the amount of tabs shown is different, depending on if the user wanted a separate rule viewing and editing screen and if they wanted it in a separate screen or in the same screen, but a different tab.

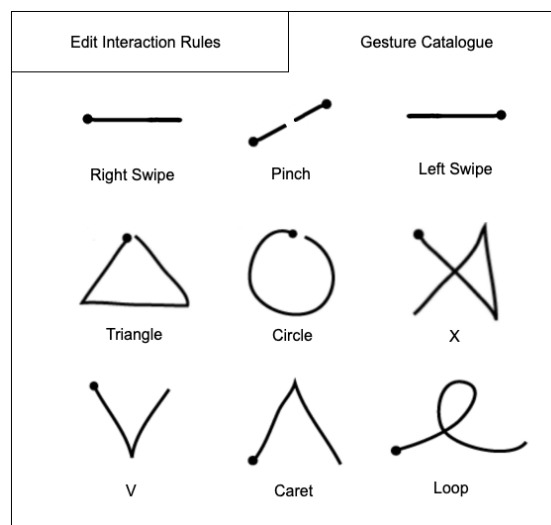


Figure 4.3: A mock-up of what the gesture catalogue could look like

The next question that is asked, asks the respondent whether they would like to have a gesture catalogue and if so, where they would want it. The question has two different sets of responses based on if the respondent wanted two different screens in two different pop-up windows, one for viewing and one for editing rules, or if they wanted a single pop-up window. If they

wanted a single pop-up window, there are three possible responses, namely: “I don’t want a gesture catalogue”, “Add it in another tab in the same pop-up window”, and “Add a second button to the application opening a separate screen with the catalogue”. If the respondent wanted two different pop-up screens, there are five possible responses and the first and last responses are the same as in the other version, except that a third button would be added if the respondent would choose the last response. The other three responses ask in which pop-up the user would like the gesture catalogue to be added, namely the “View Rules” screen, “Edit Rules” screen, or both.

Editing Gesture Interactions

For the last question of the survey, the respondent is asked what type of interface they would prefer to edit the gestures. The first possibility is to use simple drop-down menus with the name of the gesture. This does not take a lot of space in the interface, but gestures have to be recognised by their name. Another option would be to use a button that opens a pop-up window with the gesture catalogue, where the user can click on the gesture they want to use. Lastly, icons could be added in front of each gesture name in the drop-down menu, such that the user has both a more visual and textual representation of the icon. Assuming that the application contains a gesture catalogue, the user could still look up the exact shape of the gesture in the catalogue. The question then asks which of these three variations the respondent prefers.

4.2 Results

We received a total of 40 fully filled-in surveys. Of the 40 respondents, 21 identified as male, 18 as female, and 1 participant chose the “other” option, specifying they identify as “non-binary”. The average birth year of the respondents was 1992 (meaning 31 or 32 years old at the time of filling in the survey), with a standard deviation of 13.7 years. Among the respondents, 5 people reported having a high school degree as their highest achieved degree, while 15 respondents stated they have earned a bachelor’s degree, and 20 respondents have earned a master’s degree. Out of those with a higher degree (bachelor’s or master’s), 27 respondents responded that they have had at least one computer science related course during their studies, while 8 of them did not. Among the 27 respondents who took computer science-related courses, 6 respondents indicated that they did not have many computer science related courses during their studies (1 to 2 courses), while 8 respondents

have had a few computer science related courses (3 to 4), and 13 respondents responded to have had many Computer Science-related courses.

4.2.1 Extra-UI

For our first question “Would you rather have a single screen where the rules can be edited (and consequently viewed)? Or would you prefer to have separate screens, one for viewing and another one for editing these rules?”, as we can see in Figure 4.4, 28 respondents preferred the first option and 12 respondents thought a separate rule viewing screen could be useful. A clear majority of the respondents chose the option of a single screen, which means they did not see a clear advantage of a separate screen to view the rules if we can already view the rules in the rule editing screen anyway.

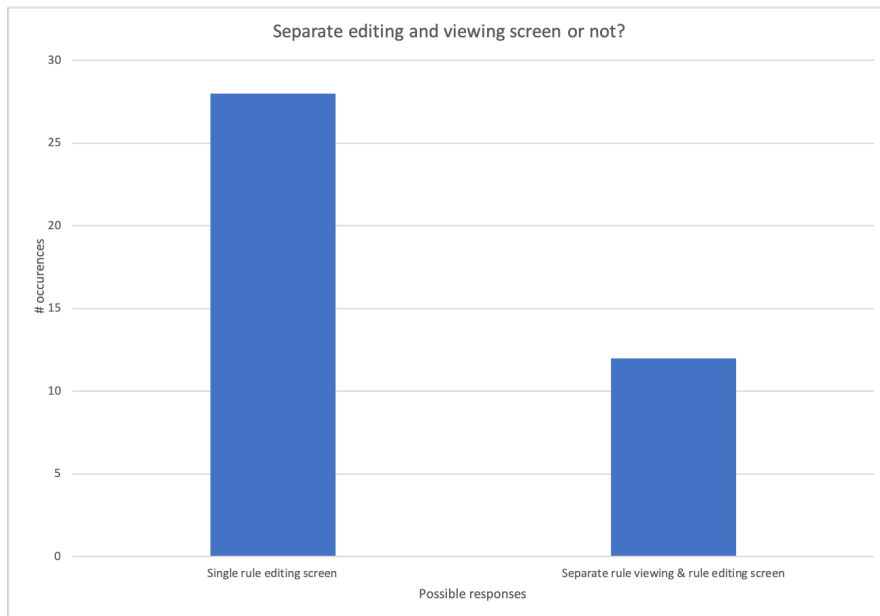


Figure 4.4: Graph of the results of the first question about the Extra-UI

For the follow up question, which was only asked if the respondent answered that they would like a separate rules viewing screen, the responses were more divided, although the smaller number of people who chose this option also means a very small sample size for this question. The question asked whether the respondent preferred to have the rule viewing and editing screens in the same pop-up in different tabs, or if they would prefer to have two separate pop-ups, and consequently two buttons in the user interface. While seven people responded they would prefer the single screen, five people responded they would like to see two separate screens.

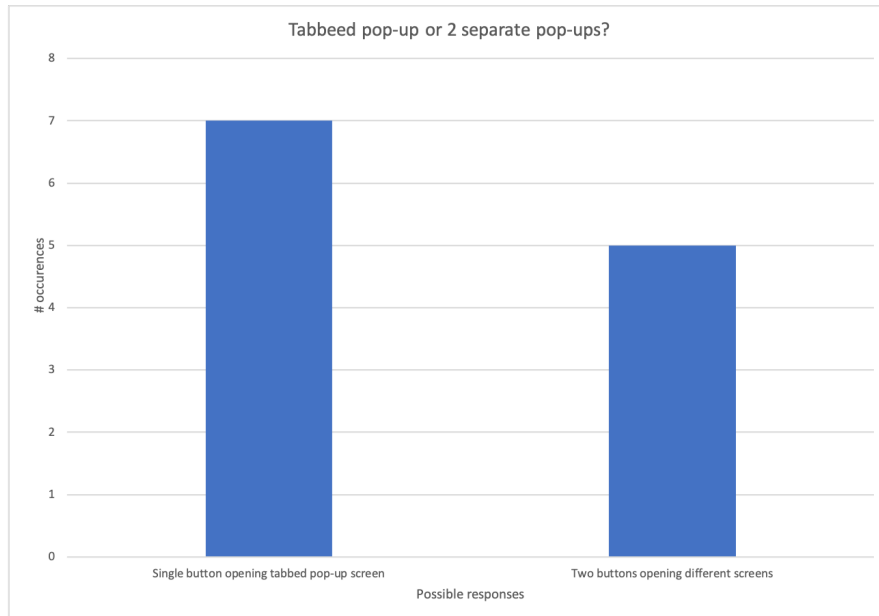


Figure 4.5: Graph of the results of the second question about the Extra-UI asking whether the user would prefer to have a single tabbed pop-up screen or two separate screens

The last question relating to the layout of the Extra-UI asked the respondent if they would like to see the types of interactions grouped into different categories, for a more structured overview of the rules, and if they would, by which property of the rules they would like to group the rules. As illustrated in Figure 4.6, five respondents responded they did not want the rules to be divided into different categories, seven responded they would like to see categories based on the type of action of the rule, 27 responded that they want to see categories based on the type of trigger of the rule, while one person responded “Other” and wrote their own response.

For this question it is very clear that people prefer the rules to be grouped based on the type of trigger that is defined for them, meaning gesture interactions would be grouped, contextual interactions would be in another group, and so on. The one “Other” response is also interesting as they suggested to implement a feature allowing the user to select which grouping they prefer. This is a good suggestion as, even though the responses are quite one-sided for this question, personal preference always plays a role in the user experience and the implementation of this feature would not be too difficult, as long as the types of both the triggers and actions have been defined.

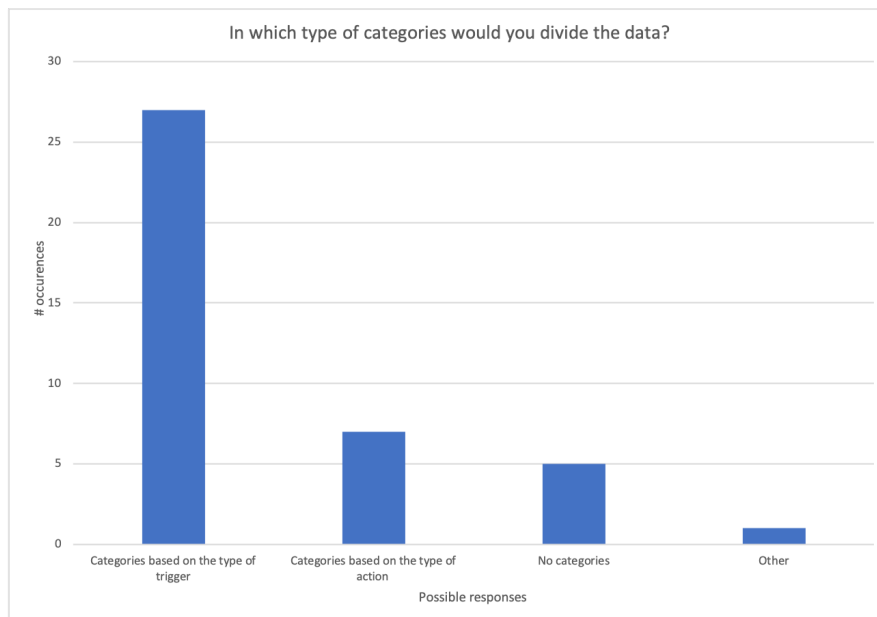


Figure 4.6: A graph of the responses to the question asking in which categories the rules should be grouped in the UI

4.2.2 Interaction Rules

In the next question of the survey, the respondents were asked which notation they would prefer for the rules. Two options were suggested and there was also an “Other” response allowing for free user input. For this question the responses were very divided, 20 respondents selected the “IF *trigger* THEN *action*” notation, 19 respondents selected the “trigger → action” notation, while one person selected “Other” and wrote their own suggestion. It is clear that the notation of the rules is very much a personal preference and we do not know the exact reason why people chose one option over the other, but for the *IF THEN* notation it was mentioned that this was the notation currently used in the *rules* view of the eSPACE authoring tool, which might have influenced the respondents into considering consistency across eSPACE as a whole over their personal preference. Lastly, the person who selected “Other” suggested to have an “WHEN trigger DO action” notation over the *IF THEN* notation. They said they think it is more intuitive to an end user than *IF THEN*, as *IF THEN* is a concept from programming languages and “WHEN trigger DO action” would be more intuitive to a person without programming knowledge.

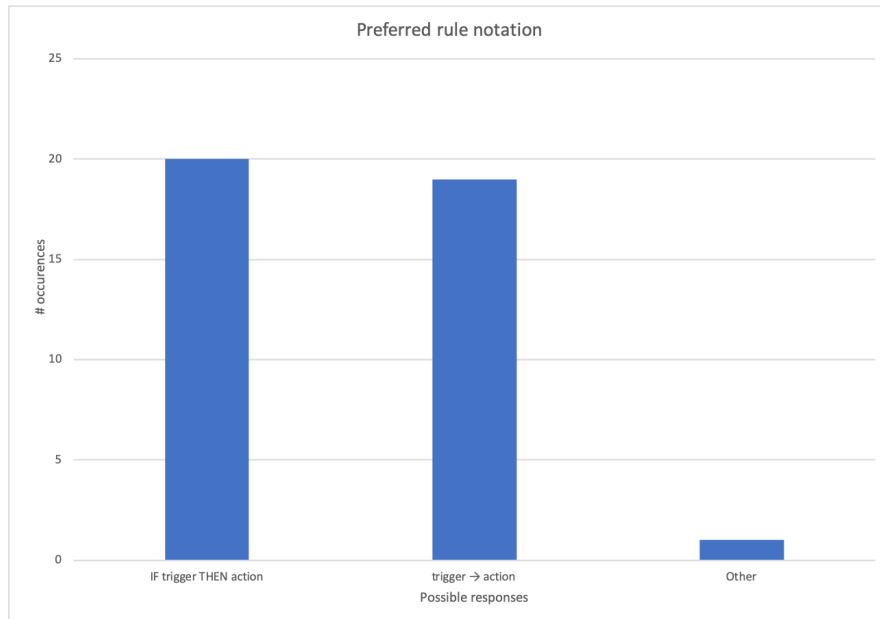


Figure 4.7: A graph of the responses to the question asking in which rule notation the respondent would prefer

4.2.3 Gesture Catalogue

In terms of the gesture catalogue, the respondents were asked whether or not they wanted a gesture catalogue, and if they did, where they would like to access this catalogue. The survey contains 2 different questions regarding this matter and the user would only get to see one of these questions, based on whether or not they chose to have a separate rule viewing screen or not. The possible responses to the questions were to either have no gesture catalogue, to add it in a tab in an existing pop-up window, and to add a separate button to the user interface. The difference between the two versions of the questions is that for the option of adding the catalogue in a different tab in an existing pop-up, the separation was made to see if the user wanted to add the tab to the rule editing screen, the rule viewing screen, or both. In Figure 4.8, the responses of the 2 questions have been aggregated.

Out of all the respondents, 2 people indicated they would not use a gesture catalogue, 20 people indicated they would add it in another tab in an existing pop-up, while 18 people would add a separate button for the gesture catalogue. Out of the people with a separate rule viewing screen, 2 chose to add the gesture catalogue to both the rule viewing and editing screens, while 1 person would like to add it to the rule viewing screen, and 2 respondents

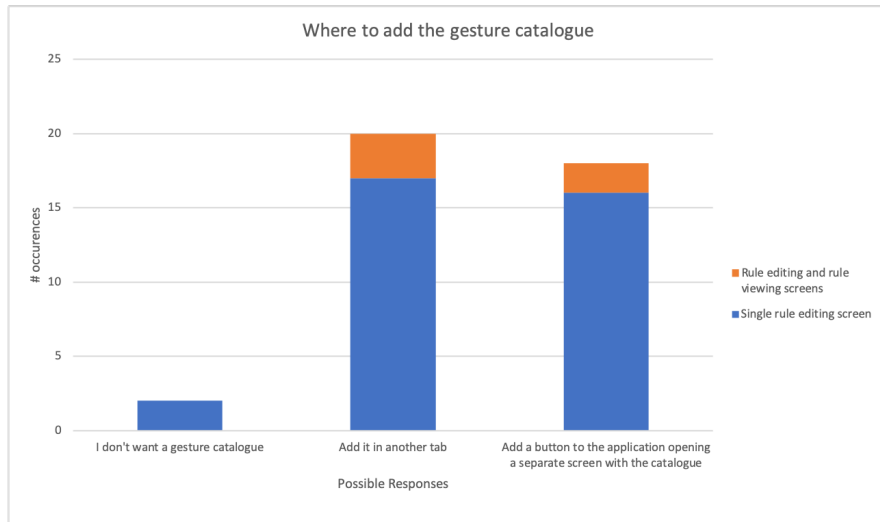


Figure 4.8: A graph of the responses to the question asking where to add the gesture catalogue

would want the gesture catalogue in a separate pop-up. These responses show a rather even distribution between the options of adding a separate button in the interface or adding the gesture catalogue in another tab in an existing screen. Again, we do not know the exact reasons for these choices, but it shows that even if a person did not want a separate pop-up for the rule viewing screen, they could still prefer to have the gesture catalogue in a separate pop-up, either for easier access while using the application, or for any other reason to choose this option.

4.2.4 Editing Gestures

For editing the gestures, 31 people responded that they would prefer to use a drop-down menu to select the gesture, of which 29 people preferred the version of the drop-down where, next to the name of the gesture, there would also be an icon indicating what the gesture looks like. The other option of selecting the gestures from a pop-up with the gesture catalogue received 9 responses. This shows that the large majority of respondents preferred the faster and easier option of the drop-down menu over the more visual approach of having the gesture catalogue pop up.

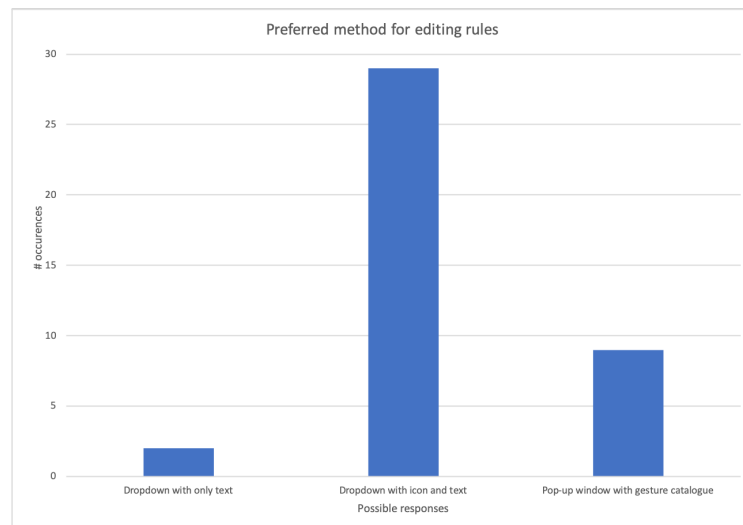


Figure 4.9: A graph of the responses to the question asking which is the respondent’s preferred method of editing gesture interaction rules

4.3 Conclusion

The results of the survey have provided some valuable insights into the preferences of the respondents regarding the design decisions for the Extra-UI. Some questions had responses that were picked by a large majority of people, while for others the distribution of preferences among two different options was almost completely evenly split. From these results, it shows that for some design choices there might be options that are clearly liked by more people than others, at least for the population of respondents to the survey, while for other design choices, personal preference might play a bigger role. From this, we can take away that these kinds of surveys can be useful in the decision making process, where the designer of the user interface should probably consider the most popular choice for the questions with a clear favourite, while for the questions that were completely divided, the designer will have to look solely at their own expertise, or the expertise of others within the field, to decide on which interface design would be best. In the case where opinions are divided, the designer could also offer both options to the user, by implementing both options and allowing the user to toggle between them, to suit their personal preference.

An example of a question where there was a rather clear favourite in terms of the responses was for example the question asking whether or not the user wanted a separate rule viewing and rule editing screen, where the “Single

rule editing screen” was preferred by 70% of the respondents. Another such example was the question on how the user would prefer to select the gesture, where 75% of the respondents preferred a drop-down menu (70% drop-down with an icon of the gesture) over the more visual approach of having the gesture catalogue pop up. For these design decisions we should strongly consider these more popular options to implement in the final implementation.

An example of an almost completely divided question is the question about the notation of the rules. Half of the respondents responded that they prefer the “IF trigger THEN action” approach, while 47.5% of the respondents preferred the arrow notation, and one respondent preferred a textual notation, but with “WHEN trigger DO action” as keywords. In this case, we have to decide for ourselves which of the options we implement, or like mentioned before, it could be a possibility to let the user configure this for themselves. Other questions that were rather split are the ones where the respondent was asked where they would like to see different screens (like the gesture catalogue and rule viewing screen) in a tab in the same pop-up, or if they wanted a separate button in the interface, which would open a separate pop-up window.

In conclusion, the survey findings provide valuable insight into the preferences of potential users. By knowing what the most popular choices are for questions with clear favourites, knowing on which questions user preference was split, and by recognising the limitations of survey data, we are able to make a more informed decision on what our Extra-UI should look like and act.

5

Implementation

The design principles and ideas that we discussed in Chapter 3 will now be applied to a real-world use case of an end-user authoring tool. First, an overview of the presented tool is given where the purpose, different mechanisms, views, and features of the tool are discussed. Second, we will present an implementation of user-defined gesture interaction in the tool, how it uses the solutions presented in Chapter 3 to solve different challenges, and how the proposed design principles have or have not been useful in the design of the implementation. Lastly, a discussion will be presented what the possible limitations, and thus also possible improvements, of the implementation might be.

5.1 eSPACE Authoring Tool

The tool we will extend with user-defined gesture interaction is the eSPACE authoring tool [26]. It is a tool focused on the design and development of cross-device and Internet of Things (IoT) applications by end users. Cross-device interactions are, like the name suggests, interactions between different devices. An example of such an interaction is for example that a person's smartphone, when it gets into Bluetooth range of that person's laptop, connects to that laptop and automatically backs up its photos to that laptop. The Internet of Things is the concept that physical objects are digitally

connected to the internet and that they contain sensors and software that measures certain statistics or allows the object to be controlled remotely.

The eSPACE authoring tool allows end users to create applications for IoT and cross-device interactions. The tool is made up of four views which aid in the creation of applications, namely the *home*, *UI design*, *interaction* and *rules* view. The UI design view allows users to design user interfaces for their applications, the *interaction* and *rule* view respectively provide a graphical and a textual way for the user to define cross-device and IoT interactions for their applications. These interactions are then saved as “rules”. The home view then provides an overview of the applications, rules, devices and other users, it further allows the user to navigate to the other views to edit the applications or rules. Lastly, there is also an *app* view which allows the user to see the final user interface for an application they created.

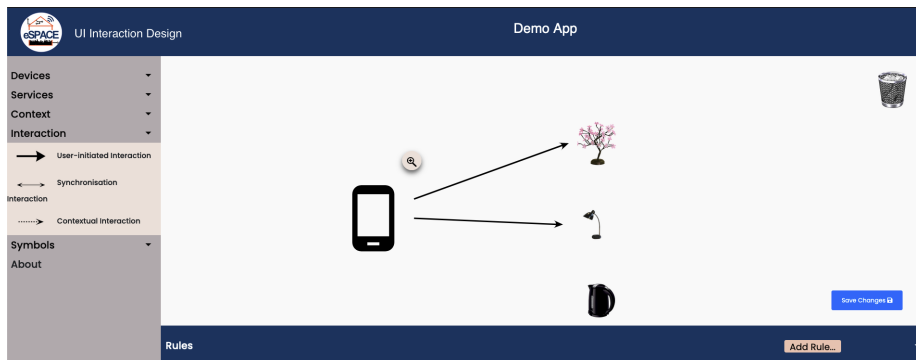


Figure 5.1: A screenshot of the interaction view of the tool

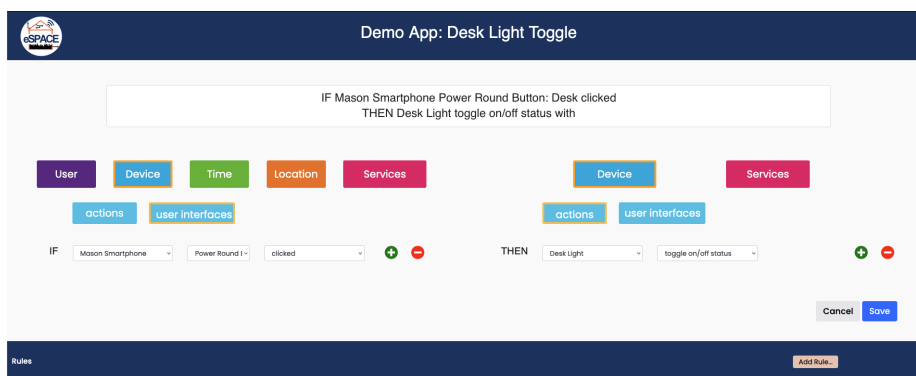


Figure 5.2: A screenshot of the rules view of the tool

5.1.1 App View

The view we will be focused on is the *app* view. Users can navigate to this view by selecting the **Open** button of an application in the list of applications in the *home* view. When opening the *app* view of an application, a new tab will be opened for each UI that has been made for this application. An application can have multiple UIs because there can be multiple devices involved in the interactions of an application, with each device having its own UI.

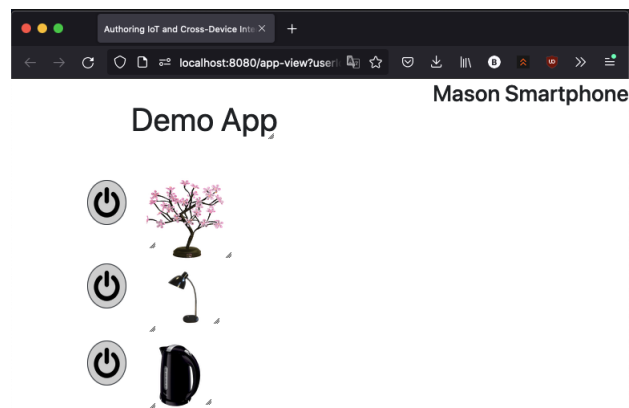


Figure 5.3: A screenshot of the *app* view showing a simple application made in the tool that can be used to turn smart objects on and off

Currently the tool provides the applications with one possible gesture interaction, which can be used in the app view. The gesture that is currently implemented is a long press on a UI element. When the user performs this gesture, a popup window will appear, asking the user if they want to distribute the selected element to another device which currently has a running instance of the eSPACE authoring tool. Once the target device selected, the UI element will be added to the application that is currently running on this device. This is currently the only gesture interaction that is possible in the applications created by the tool, besides the standard *on click/press* event, which triggers an interaction when a UI element is clicked.

The *app* view is the view to which we will work to implement user-defined gesture interaction. Not only will the user be able to trigger gesture interactions in this view, the Extra-UI containing the list of the defined gesture interactions, together with the gesture catalogue will also be accessible from

this view. In this Extra-UI, the users will not only be able to view, but also add, delete, and edit the gesture interactions of the application.

5.2 Implementing Gesture Interaction

In this section, the implementation of the use case will be discussed. This means that we will discuss the possible options that exist for integrating user-defined gesture interaction into the eSPACE authoring tool and we will present the actual implementation that was made. Different possible design decisions will be discussed, after which each of the options will be evaluated against the principles and ideas discussed in Chapter 3. The goal is to find the best possible ways of implementing gesture interaction for this specific use case, which is the eSPACE authoring tool and the goals it tries to achieve. Multiple theoretical possibilities are discussed for each of the design decisions will be discussed, after which we will present how we handled the actual implementation, as decisions have been made for the feasibility of the implementation in the context of this research.

5.2.1 Gesture Support

Because the eSPACE authoring tool is used to create cross-device and IoT applications, in terms of supported gestures, it could make sense to allow different kinds of gestures for each of the different kinds of devices. For example, touch gestures could be used on devices with touch screens, mid-air gestures could be used on devices that can detect mid-air gestures, like for example devices that contain an accelerometer, or the Soli radar sensor [13], which was discussed in the chapter on related work. This allows for complex and very personalised gesture interactions with the user's devices, but the obvious limitation here is that a gesture set for each of these different types of gestures has to be created, together with a recogniser that accurately recognises these gestures. With how many different devices exist, both smart devices and things, it is practically impossible to do this for such a large amount of devices. It would be possible to create gestures for a few specific devices and technologies, which would still limit the use to these specific use cases, but this could be enough depending on the goal we are trying to achieve with these gesture interactions. One solution would be to crowdsource the gesture sets and/or recognition for different devices by allowing people to create and upload their gesture implementations for a certain device. This approach brings a whole new set of challenges with it.

The other option is to support only a single kind, or a few kinds, of gestures which are well-defined and implemented for a specific technology. The most obvious kind of gestures to implement are touch gestures as the applications use a screen interface on the devices, which often have touch screen technology, but mid-air gestures could definitely also be used. This approach is more realistic than the previous one, but it does mean that not every device might have a gesture interaction interface, even though they might have another way to detect gestures. In practise, only implementing one or a few types of gesture interfaces could already improve the workflow of the application.

This is the reason why, in our implementation, we chose to focus on touch gestures. The applications created using eSPACE all have a screen interface, and for many devices this screen will also be a touch screen, allowing the implementation to be used on as many devices as possible. The gesture recogniser we used in the implementation can even recognise gestures performed using a computer mouse, which means the gestures can be performed on virtually any device. Besides the aforementioned advantage, touch gestures are also the most common type of gestures for which libraries with recognisers can be found. As our research is not focused on the technology behind defining and recognising different gestures, we will use one of these recognisers, and more choice of libraries means more chance of finding a recogniser suited for our preferred criteria.

5.2.2 Incentive

The advantages and incentives for using gesture interactions in the applications that were discussed in Chapter 3 still apply to this use case. A first advantage is saving space in the UI such that it is less cluttered, which makes it easier to find UI elements. Especially in the current version of eSPACE, this might be important as it currently only supports a single static view per device for each application, which means that an application can get cluttered rather quickly. Of course there are only a certain amount of gestures that can be implemented per detection method (touch, camera, etc.), which means this amount will not be too large, but it is not negligible, especially in the current eSPACE implementation. Another advantage is that it can improve the speed of the user's workflow. For mid-air gestures this improvement can be made by removing the need to even touch the device for certain interactions. Some gestures might also be more intuitive to perform to some users than interacting with the UI elements of the application. This could speed up the workflow, but even if the speedup in the user's workflow

is negligible or none, the intuitiveness might also be an advantage in itself, as using a more intuitive way of controlling the application can increase the user satisfaction when using the applications. As the eSPACE application is mainly focused on users who create applications for their own devices, the interactions they design for themselves should be more intuitive to them, so giving them more customisation options in the form of user-defined gesture interaction should only improve this.

5.2.3 Location of Definition

Another consideration to make is where in the tool the gesture interaction will be defined. One option is to create a new view in the tool with the sole purpose of creating gesture interaction for applications. This is an option, but as the tool already has two views for designing interactions, namely the *interaction* and *rules* view, integrating the definition of user-defined gesture interaction into these views seems like the more apparent approach. Both approaches have advantages and disadvantages, but integrating the gesture interaction into the existing views is consistent with the current paradigm of the tool. The gestures will be the trigger of the trigger-action rules created to define the gesture interaction and in the *interaction* view, the possible gestures could be added to a separate dropdown in the sidebar of the view. Lastly, the gesture interaction could also be implemented into the UI view, but this would be inconsistent with the current idea of separating the UI design and functional interactions in the tool.

We have also decided to allow the users to edit, create and delete defined gesture interactions in the Extra-UI that will be added in the *app* view. The Extra-UI will already contain these interactions as information to the user, as well as the gesture catalogue, which shows the set of possible gestures. In terms of usability, it would be much easier for the user to edit the gestures in the *app* view if they would be using the application and wanted to change a gesture. Instead of navigating to the *interaction* or *rules view*, the user would be able to change the gestures directly from the Extra-UI of the application, as navigating to one of the former views could even mean having to use a different device, as the *app* view is available on any device, but the other views are not.

5.2.4 Interaction Levels

What is important to consider next are the levels of the application the gesture will interact with. Because currently the applications created using the eSPACE authoring tool are single view applications, the only levels of interaction are application level interactions, element level interactions and device level interactions. It seems quite straightforward that application level gesture interactions are a good idea, because applications are created to control smart device and IoT interactions and the gestures are just another mode of interaction. Because the application can or will be ran on multiple devices, it should be considered that the application level gestures can be performed through a technology that is shared between most devices, because not every device can recognise the same type of gestures. That is why touch gestures seem like a good choice for this kind of gestures, as the touch screens of different smart devices can be used to perform these gestures. An important detail that has to be considered when designing the gesture set is that different devices might have a built in device level gesture set that the operating system provides to control different things on the device, which means clashes with these gesture sets could occur if not chosen carefully.

If we would implement different gestures for different technologies it seems like the option of having device level gestures for a certain application would be a good idea. Certain interactions could then be triggered by interacting with a certain device, with the gestures that are implemented for the sensors of that device. This could make things more complicated for the user as it introduces quite a lot of different options, which is a good thing for customisation, but it can reduce the clarity of which gesture interactions are possible in a certain application.

Lastly, there are already element level gesture interactions defined in the current version of the tool. The first one is the *on click* event which triggers a customisable interaction when a UI element is clicked. Second, there is the long press on a UI element, which is not customisable, and which allows the user to send the element to a different device. Adding user-defined UI element level gesture interaction seems unnecessary for the purpose of the eSPACE authoring tool, as the focus lies on cross-device and IoT interactions, not on the UI of the application. Even if we would want to implement element level gesture interaction, first we would have to find what interactions would be possible besides the currently existing action of *sending a UI element to another device*. Secondly, it is hard to think of gestures that can be performed without having clashes with other levels of gesture interaction,

or even just clashes with interactions that exist for the UI elements. Only when gestures are found that are distinct enough from others and when multiple incentives for including them can be found, then we can only start to let the users define their own element level interactions.

5.2.5 Clarity

In terms of clarity of gesture interactions, the user creates their own gesture interaction, so this might be less important than it is for other use cases, but it stays important to keep this aspect in mind. Not only because users can share their applications with others, but also because users might forget the interactions they have defined. A good gesture vocabulary and the fact that the gesture interaction is user-defined might help to remember it, but this is not a guarantee. If gestures are designed using the *interaction* and *rule* view, the user can check these views of the tool to see which gestures are defined. Realistically, it is not desirable and convenient to check the tool if a user wants to know which gesture interactions exist, which is why some help can be implemented. Earlier we have discussed options of doing this through either demonstration or using an Extra-UI.

Like we already mentioned, showing gesture interactions through demonstration is hard because the demonstration has to be created and it has to be shown at the right time to the user, but as the user can define their own gesture interaction both of these things are hard to accomplish. Showing the interactions of the user through text or icons seems like an easier and sometimes clearer way of describing the interactions. For this we could create an Extra-UI which can be triggered by a small button in the interface, and which would list the gesture interactions that are possible in this application, or for this device. This way the user can easily pull up the defined gestures of the application. The tool could ask the user to write a description for each gesture interaction when they are created to show them here, or the interactions could be shown in the same rule format as in which they are defined.

5.2.6 Safety

Lastly we can explain the measures that can be implemented to improve the safety principle which was discussed in the previous chapter. One way to minimise the amount of times gestures get accidentally triggered is by

designing a good gesture vocabulary. Some factors that indicate a good gesture vocabulary have been discussed earlier. Factors like the intuitiveness and consistency of the gestures let the user remember the gestures more easily and possibly perform them more accurately. Distinctiveness between gestures can also help with the recall of the user, but it will also help when it comes to the accuracy of the gesture recognition algorithm.

All these factors and more will help at preventing accidentally triggering a gesture, but the possibility of it happening still exists and it is probable that accidents will happen. If a gesture triggers accidentally, it might be an important factor to consider whether or not the action triggered by the gesture is deemed critical or not. If the action is not deemed critical and is easily reversible, the accidental trigger might not have the greatest impact and can just be reversed by the user, but if this is not the case, it would be desirable to have some system in place to prevent the effect from happening in case of an accidental trigger. A common way to add an extra safety layer is to ask for confirmation in a pop-up window when the user is performing a critical action. This greatly decreases the chance of a critical action accidentally happening, although it is still possible.

The design choice now lies in how to let the end user create these confirmation pop-ups when designing their own gesture interactions. We could force the user to create a pop-up for every gestural interaction they design, but this is generally not desirable as in some cases the incentive for gestures is a speed up in the workflow, or in the case of mid-air gestures it could be the fact that the user has no need to touch the device. These incentives are both reduced or even nullified by the introduction of pop-up windows, because it slows down the workflow with an extra action and introduces a need to touch the device. Some technologies could allow for easy confirmation, for example if the mid-air gestures are performed by a controller which has a button, pressing the button could be used as a second confirmation of the intent of the action. If the mid-air gestures are performed by using the accelerometer of for example a smartphone device, it could be easy to confirm the action on the screen. For our implementation of user-defined gesture interaction in the eSPACE authoring tool, it seems better to allow the user to create pop-up windows, but not to enforce it for every gesture. Especially due to the possible variation in devices, it would be hard to create a universal solution for the aforementioned problem. As eSPACE applications are targeted towards personal use, in the end it might be desirable that, the user can choose their own preferences, the application should just guide them to a better design.

A challenge still lies in how to advise the user to create a pop-up for critical actions, which could for example be achieved by creating a tooltip explaining the use or function of the pop-up window.

5.3 Practical Implementation Details

In this section we will discuss the technical details of our implementation, what that implementation looks like and how it behaves. First, the gesture recogniser used in the implementation will be presented, along with why it was chosen over other options. After that, we will present the implementation of the Extra-UI in which the users can see the defined gesture interactions and edit these interactions.

5.3.1 Gesture Recognition Library

The eSPACE authoring tool is a web app that uses a Spring Boot backend that serves HTML pages with CSS and JavaScript. No frontend framework is being used, which means that for easy integration into the current implementation, we would need a plain JavaScript implementation of a gesture recogniser.

Quite a number of gesture recogniser implementations are available and our main focus when selecting a recogniser to use was the amount and variation of gestures that are available. As we discussed in the previous chapters, gesture recognition accuracy is influenced by the amount of gestures being recognised, but also by the similarity between the gestures. Even if the recogniser is proficient at differentiating between similar gestures, the gesture vocabulary also has to be clear to the users of the application. This might be affected by the similarity of gestures, the user's familiarity with the gestures and the amount of gestures. In Table 5.1 you can find a list of gesture recognisers with a JavaScript implementation that could possibly be integrated into the eSPACE authoring tool. Each entry in the table lists the supported gestures, extra features and the URL to the home page of the website of the library. The first entry in the table, namely the “\$1 Unistroke Recognizer” is the recogniser we integrated and used in our implementation.

	Supported Gestures	Notes	URLs
\$1 Unistroke Recognizer	Triangle, X, rectangle, circle, check, caret, zig-zag, arrow, left square bracket, right square bracket, V, delete, left curly brace, right curly brace, star, pigtail	Allows custom gesture definition	http://depts.washington.edu/acelab/proj/dollar/index.html
HammerJS	tap, doubletap, press, horizontal pan, swipe, pinch, rotate	Pan and swipe in multiple directions can be activated	https://hammerjs.github.io/getting-started/
TouchSwipe	swipe (up, down, left, right), pinch (in, out), single or double finger touch events	Supports click events on objects, definable threshold of when a swipe is recognised	http://labs.rampinteractive.co.uk/touchSwipe/demos/index.html
interact.js	tap, doubletap, hold, rotate, pinch, drag		https://interactjs.io/
ZingTouch	tap, rotate, pinch, expand, pan, swipe		https://zingchart.github.io/zingtouch/
QuoJS	tap, doubletap, hold, 2-finger tap, 2-finger doubletap, swipe (up, down, right, left), swipe, drag, rotate (left, right), rotate, pinch out, pinch		https://github.com/soyjavi/QuoJS
Alloyfinger	tap, doubletap, long press, single tap, rotate, pinch, swipe	Has a “pressMove” event that returns the travelled distance	https://github.com/AlloyTeam/AlloyFinger

Table 5.1: The different gesture recognisers considered for the implementation along with the gestures they support and the URL to the homepage of their website

The Selected Recogniser

The \$1 Unistroke Recognizer is a single-stroke gesture recogniser. This means that, like the name suggests, it recognises the shape of touch gestures that are made using a single stroke on the screen. This is also immediately the most limiting factor of the recogniser. Being only a single-stroke recogniser means that gestures using two or more touch points on the screen cannot be recognised by the recogniser. This means that gestures like a two-finger touch or a pinch gesture, that can be recognised by some of the other recognisers in the list, cannot be recognised by the \$1 Unistroke Recognizer, which at first does not seem like an insignificant limitation.

For the purpose of our implementation, this limitation is not as significant as it might seem, mainly because of the cross-platform characteristics of the eSPACE authoring tool. First of all, many smart devices reserve more complex multi-touch gestures for their operating system features, like for example performing a three-finger swipe up or down on certain versions of Android takes a screenshots of the selected area¹. As mentioned in Chapter 3, it can be important to avoid clashes with other gestures to prevent the user accidentally triggering a gesture interaction they did not intend, and for this purpose, the single-stroke gestures might be our safest option. Not only are single-stroke gestures an easy solution to the possibility of clashing with predefined gestures from the operating system, like mentioned before in the previous section, these gestures can be performed using a computer mouse. This means that any device which uses a mouse pointer in its interface can perform the gestures defined by our implementation, which means not only touch gesture devices, but many other screen devices have a way of performing these gestures.

The other advantage of the \$1 Gesture Recognizer is that it allows us to define our own gestures. The recogniser recognises the performed gesture by comparing the shape of the performed gesture to the shapes of the defined gestures, which are defined as a single “template” gesture which is a set of points tracing the stroke of the gesture. This means that gestures can be easily defined by recording a stroke on the screen and adding its set of points to the list of defined gestures. It uses a nearest-neighbour approach with a Euclidean distance function to calculate the similarity between the points defined by the performed gesture, in comparison to the points defined by the predefined gestures. It uses techniques like resampling the points, rotating

¹<https://c.realm.com/in/post-details/1248444623907454976>

and scaling the shape to account for respectively differences in speed, angle and size of the gestures. For more details about the implementation and the mathematical foundation of the recogniser please refer to the paper written by the creators of this implementation [30].

5.3.2 The Gesture Vocabulary

Ultimately, we did not define any custom gestures, as the implementation of the recogniser on the website², where a demo can be found, already includes a defined set of 16 gestures, which were used in the research done by the creators of this implementation. From this set of 16 gestures, which can be seen in Figure 5.4 we selected 8 gestures which are distinct enough for an optimal gesture recognition performance, and which are, subjectively the easiest gestures to perform out of the set of 16, without too many complex shapes in the stroke. The gestures used in our implementation are the *triangle*, *x*, *circle*, *caret*, *left square bracket*, *right square bracket*, *v* and *pigtail*. We renamed the *left square bracket* and *right square bracket* to *left bracket* and *right bracket*, as these were the only brackets in our set of gestures, and the *pigtail* gesture was renamed to *loop*.

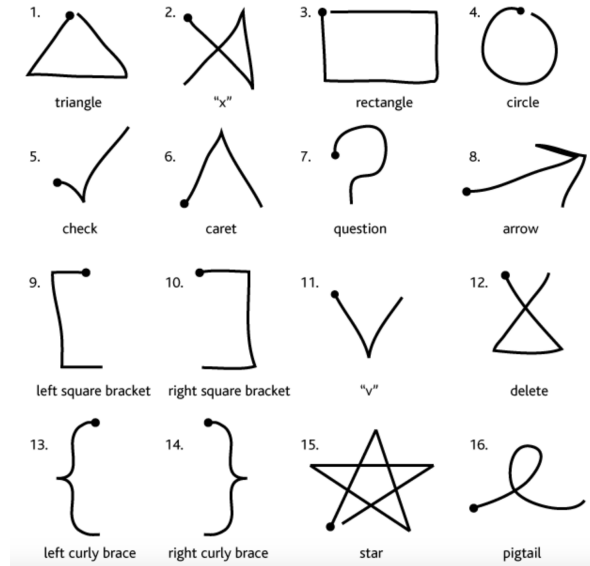


Figure 5.4: The set of 16 predefined gestures defined in the \$1 Unistroke Recognizer implementation

²<http://depts.washington.edu/acelab/proj/dollar/index.html>

5.3.3 Implementing Gesture Interaction

The original implementation of the \$1 Unistroke Recognizer defines the predefined gestures as a list of “Unistroke” objects which have a name and a list of points describing the shape of the corresponding gesture. For our implementation, we have removed 8 gestures from the original list of predefined gestures and renamed some of the remaining gestures, such that we are left with the set of gestures mentioned earlier. When the user performs a gesture, it will be compared to these 8 predefined gestures and the closest match, together with its similarity score will be returned. This modified version of the gesture recogniser has been added to the code base of the eSPACE tool.

The gestures performed to trigger the interactions defined by the user are performed in the app view of eSPACE. Consequently, the code of the app view (in the app-view.html file) has been modified to recognise gestures performed in this view. First of all, the area in which gestures should be recognised is defined, which is in this case the entire body of the HTML document. After this, multiple mouse or touch events are registered on this area, to record any strokes made inside this area and possibly recognise them as one of the defined gestures. Whenever the user clicks their mouse, or touches their touchscreen inside the application, a list will be created in which coordinates of points along the stroke of the performed gesture will be stored temporarily. As long as the mouse click or touch is being held, every movement will be recorded, and when the mouse or touch is released, the performed gesture will be compared to the list of predefined gestures by the gesture recogniser. A gesture will only be recognised if at least 10 points have been recorded, as to avoid false positives on simple clicks or small accidental movements.

When an existing gesture has been performed, the application will check if any interaction has been defined for this gesture. Gesture interactions are defined as objects, like the other defined interactions in the application, and have 3 properties, namely **gesture** which contains the name of the gesture, **uiElement** which contains an object referring to the UI element on which the gesture needs to be performed to trigger the interaction, and **action** which contains an object corresponding to the predefined action that will be triggered when the gesture has been performed. If a gesture has been performed and an interaction exists for this gesture, the defined action will be triggered. The option of showing a message that tells the user which gesture has been performed, and which action that triggered, has also been added to the application. A so-called “snackbar” message with this information will

be shown to provide the user with more clarity about when a gesture, and which gesture, has been triggered. In Figure 5.5 an example of a snackbar message is visible that would pop up when the user performs the `Triangle` gesture, which would toggle the `Tree Light Toggle` action.

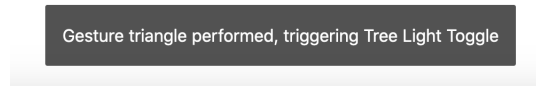


Figure 5.5: An example of a snackbar message that would pop up at the bottom of the application after a gesture was performed

5.3.4 Implementing the Extra-UI

In the current eSPACE application, the interactions of any applications created using the tool are defined in the interaction view or rules view. Earlier we already discussed the idea of adding an Extra-UI displaying the defined interactions of the application, but not only can the Extra-UI be used for displaying the defined interaction rules, it could also be used to edit them. In the survey, discussed in Chapter 4, respondents were asked whether or not they would like to view and edit rules in separate screens, or if they would prefer to have one screen in which the rules could be edited and consequently viewed. A majority of respondents preferred to have a single screen where the rules can be edited, and this is also what we have implemented. The current implementation shows only the gesture interactions and allows editing of the gesture interactions, but not of all the other types of interactions, as these are outside of the scope of this research.

Viewing and Editing Rules

For the creation of the Extra-UI, a modal was created using the Bootstrap³ library. The modal is defined in the HTML of the document and is also displayed by showing its HTML on top of the rest of the application. This modal can be opened by clicking a button on the top left of the application. When the modal opens, all of the currently defined gesture interactions of the current application will be shown in the modal. When the modal for editing gesture interactions is open, none of the defined gesture interactions can be triggered, to prevent accidental triggers of any gesture interactions

³<https://getbootstrap.com/docs/4.1/getting-started/introduction/>

when using the modal to edit, add or remove any interactions. The defined interaction rules are shown in a vertical list where each line represents 1 interaction. Every defined gesture interaction is shown as a trigger-action rule written as “IF gesture: *name_of_gesture* on: *name_of_UI_element* THEN *name_of_action*”, where the names of the gesture, and UI element it should be performed on, specify the trigger of the rule and the name of the action specifies the action that gets triggered.

The screenshot displays the 'Gesture Rules' tab in the Extra-UI. It features two rows of rules, each with a red 'X' icon for removal. The first rule is 'IF gesture: Triangle on: Page THEN Tree Light Toggle'. The second rule is 'IF gesture: Circle on: Image: bureauLight.png THEN Desk Light Toggle'. A green 'Add' button is positioned below the rules. At the bottom right, there are 'Close' and 'Save' buttons.

Figure 5.6: A screenshot of the rule editing tab in the Extra-UI

Each of these lines is a custom HTML component with a drop-down menu to select the desired gesture, UI element and action for the corresponding gesture interaction rule. Using a drop-down menu to select the desired gesture is preferred by the respondents of the conducted survey, discussed in Chapter 4. The list of gestures is the list of possible gestures in our gesture vocabulary, while the list of UI elements and actions are retrieved from the database, where the UI elements and possible actions for the application are stored. Note that for the UI elements, there is one extra option called **Page**, which when selected, triggers the gesture interaction if the gesture is performed anywhere inside the application, instead of only when the gesture is performed on a specific UI element. Not only can existing interaction rules be edited by changing the selection of any of the drop-down menus of the corresponding rule, new rules be added using the green **Add** button at the bottom of the list, that will add a new empty component in which a new gesture interaction rule can be defined. A rule can be removed by clicking the red “x” at the end of the line corresponding to that rule, after which that line will be removed from the UI. The changes that the user has made to the interaction rules will only be saved after the “Save” button has been clicked. Any incomplete or empty rules will be ignored when the user saves the interaction rules.

Gesture Catalogue

In the survey, a proposition was made to add a “gesture catalogue” to the Extra-UI which would be a page that shows all the possible gestures that can be performed using an image of the shape of the gesture, together with the name of the gesture. This feature could then be used to check which gestures can be performed, but also what these gestures look like, as a user might not know exactly how to perform a certain gesture, based on the name of the gesture alone. Out of the respondents, only two people indicated they did not want or need a gesture catalogue, and out of the people who did, the opinion of where to show the gesture catalogue was almost evenly split between the given options. The options being to either add the gesture catalogue in a separate pop-up window, opened by clicking another button in the application’s interface, or to add the gesture catalogue in the same pop-up window as the rule editing screen, but having both in different tabs of this window.

Because the preferences of the survey respondents were very much split between the two options, we have decided to go for the option where the gesture catalogue is added in the same pop-up window as the gesture editing screen. This choice was made, because having one button less in the interface of the application saves possibly valuable space in the interface, but more importantly, if the user wants to consult the gesture catalogue while editing the gesture interaction rules, they can do so while not yet having saved the rules. This means the catalogue can be consulted when in the process of editing the rules, while, if the gesture catalogue was in a different window, either this window should be accessible from the rule editing screen and should be opened on top of the rule editing screen, or the user would have to close the rule editing screen, possibly losing some of the progress of the rules they have edited, or temporarily saving a possibly incomplete set of rules, before being able to consult the gesture catalogue.

5.4 Evaluation

Possible advantages and disadvantages of the design decisions have been discussed while going over the possible implementations of user-defined gesture interaction in the eSPACE application, but to know whether or not the right decisions have been made, we have evaluated our implementation with potential users to assess the usability of the system. This evaluation study was

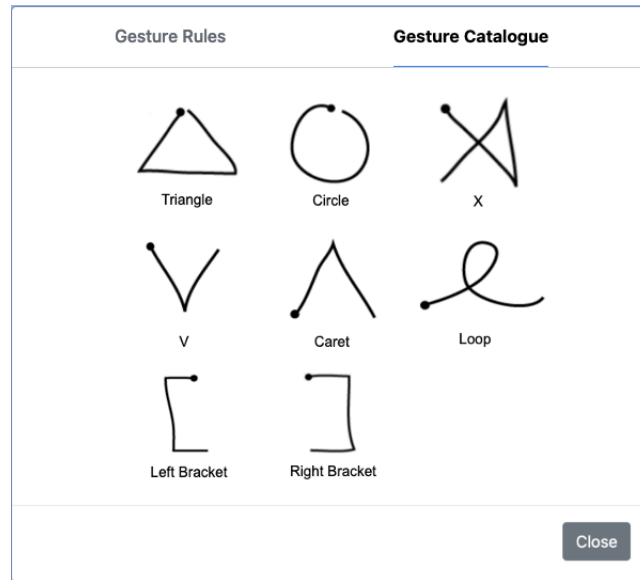


Figure 5.7: A screenshot of the gesture catalogue tab in the Extra-UI

completed by five participants, of which three were female and two were male. The female participants were 22, 25, and 53 years old, while the male participants were 24 and 54 years old. The participants first received a brief explanation of what our extension to eSPACE allows the users to do, and which screens or features there are, without demonstration of those features. They also received the information that each of the gestures was a single uninterrupted stroke and that gesture interactions could either be targeted on a certain UI element, or be available for use on the entire page of the application. Then, the participants were asked to perform certain tasks in the application according to different scenarios, of which the speed in which they executed the tasks was measured. After they completed all of the tasks, the participants were asked to fill in a Post-Study System Usability Questionnaire (PSSUQ). Besides the PSSUQ which quantifies the usability of the system according to the responses of the participants, during the whole process we also allowed qualitative evaluation from the participants in the form of comments and suggestions.

5.4.1 Tasks

After a brief overview of the system and the purpose of our implementation, the participants were told to complete a set of tasks to test the system. A simple application with three buttons and images was used, and one gesture interaction was already defined, namely an interaction triggered by perform-

ing the **Triangle** gesture anywhere on the page of the application, which would trigger the **Desk Light Toggle** action, which would toggle a desk light on or off. Because there was no connection to an actual desk light, the option to show a snackbar message was turned on, indicating which gesture was used and which action was triggered, after triggering a gesture interaction. For each of the tasks, the participant started from the app view of the application and the time they took to complete each of the tasks was measured. Each of the tasks given to the participants is listed below in the order they were completed.

- Task 1: “One gesture interaction is already defined. Explain which gesture needs to be performed and what action will get triggered by performing the gesture”
- Task 2: “Change this interaction to use the **Circle** gesture and then trigger the interaction in the application”
- Task 3: “Add an interaction that uses any gesture, besides the **Triangle** or **Circle** gesture, on the image of the water boiler to toggle the boiler on or off. Trigger the interaction in the application and after that, remove this interaction again”

Task 1

For the first task, none of the participants struggled to find the rule by opening the Extra-UI. The main difference in completion time for this task between the participants was due to the speed at which they interpreted the rule written on screen and translated it to a sentence. The lowest time to complete the task was 11s, while the highest time was 20s, and the average time was 16s.

Task 2

Then, for the second task, the users started from the app screen again and they had to change the gesture of the existing interaction to the **Circle** gesture, save their change and trigger the interaction in the app screen. For four of the participants, this task also went rather smoothly, but one participant struggled completing the task due to how our gesture recogniser works. Changing the gesture to the **Circle** gesture and saving the interaction was easy for all the participants, this took between four and seven seconds, but one participant struggled to trigger the gesture in the app because they were

drawing their circle in the wrong direction.

For the \$1 Unistroke Recognizer, the angle at which a gesture is performed does not matter, but what does matter is the direction of the stroke. Because we only had one **Unistroke** defined for this circle gesture, the circle would only be recognised when performed in one direction, namely the anticlockwise direction in which it was defined. We got rather lucky with this test, because four participants did the gesture in the anticlockwise direction, but who knows how many people would make the same mistake when using the application. The participant eventually found the direction in which the gesture had to be performed using the gesture catalogue, as it shows where the gesture starts with a dot at the starting point of the stroke, but it took them 52 seconds to complete the task, while the other participants took between six and ten seconds.

This issue is an obvious flaw of the current system, but it is also very easy to fix. The only thing we have to do is to add another **Unistroke** to the application describing a clockwise circle gesture and also recognise it as the same **Circle** gesture. Another option is to make the gestures, or their names, more specific. For example, we could have a separate **Clockwise Circle** and **Anticlockwise Circle** gesture. The same issue might arise for the **Triangle** gesture for example, but the same solutions can be used. Other gestures might not be as ambiguous as long as you draw the gesture from left to right, and top to bottom. For example the **V** gesture would in this case probably be performed correctly by anyone drawing from left to right, but as we already discussed in earlier chapters, what is intuitive has not only a personal, but also a cultural attribute. There are languages and cultures where people read and write from right to left, which means the gestures we defined might not be performed correctly. This means that, although the user has the gesture catalogue to check how the gesture should be performed, implementing either of the proposed solutions might be an improvement to the system.

Task 3

On the last task, each of the participants completed the task without any issues. Four people used the **X** gesture for the rule they created, while one person chose the **V** gesture. The four people who chose the **X** gesture also looked at the gesture catalogue to see how the gesture had to be performed in a single stroke, while the person who chose the **V** gesture did not look at the gesture catalogue. When asked why the participants chose the gesture

they did, each of them who chose the X gesture said it was because it was the first gesture in the list they were allowed to use, as it was the second gesture in the drop-down, but the first gesture was the **Triangle** gesture which they were not allowed to choose. The participant who chose the V gesture said that it was the first gesture he saw, and that he was also certain about how to perform the gesture, because “it’s a really simple gesture”. The completion times were between 19s and 30s, with the fastest time being performed by the person who chose the V gesture. One participant also commented that it was really useful to have the gesture catalogue as a feature, but that while some gestures were easy to perform, it might be good to add recognisable gestures like a simple **Swipe** or **Two-Finger Swipe** gesture. While the first one can definitely be added, the second one cannot, as this is a limitation of using a single-stroke gesture recogniser

5.4.2 Post-Study System Usability Questionnaire

After completing the tasks, the participants were asked to fill in a Post-Study System Usability Questionnaire (PSSUQ), which assesses the usability of the system using 16 statements which the participant has to rate on a 7-point Likert Scale. The overall score (average of all statements) was 2.04, while the System Usefulness (SYSUSE) score was 1.73, the Information Quality (INFOQUAL) score was 2.48, and the Interface Quality (INTERQUAL) score was 2.00. Overall it seems like the system scores well on the PSSUQ, with the SYSUSE being the best score and the INFOQUAL scoring the lowest. In terms of the System Usefulness questions, there were no negative responses (meaning from “slightly disagree” to “strongly disagree”), but the participant who struggled with the directionality of the **Circle** gesture mentioned that they responded “slightly agree” to some of these questions because of the issues they encountered.

For the Information Quality questions, while some participants were perfectly happy with the information, two others said that there could be a bit more information. The participants indicated that the gesture catalogue was a very useful feature, at least to learn which gestures there are and what they look like, and one participant also mentioned they liked the snackbar message that pops up when a gesture is performed. They mentioned it would be useful in an IoT setting where you might not be able to see any visual effect of an action you performed.

Lastly, the Interface Quality questions all got positive responses, with no extra comments besides one participant mentioning that having the gesture

catalogue in the same pop-up as the screen to edit gestures is nice when editing gestures, but when using the applications, they would prefer to also have it in a separate pop-up for a faster lookup of gestures.

6

Conclusion and Future Work

Introducing user-defined gesture interaction allows for another level of customisation in end-user authoring tools. The user can define interactions that are more intuitive to them and can in this way improve their workflow in the application and their satisfaction as a whole. Although it comes with potential, certain challenges arise when allowing a user to define their own gesture interactions. Either because this user might not have any knowledge about gestures and how they they are recognised by the system, or because of some inherent disadvantages of using a gestural user interface. Working with a fixed set of gestures and not allowing the user to create custom gestures allows us to focus on the problems that may arise with defining and using the interactions, instead of focusing on the creation and recognition of the gestures.

In this thesis, we have discussed related work that identifies challenges that occur with regular, predefined gesture interaction and the solutions that were proposed and used. From defining a gesture vocabulary with gestures that are easy to be recognised by the system, to gestures that are metaphorically and iconically logical towards their functionality. Next, we identified different challenges that occur when trying to implement user-defined gesture interaction in an easy-to-use and intuitive manner, that also guides the user towards creating better gesture interactions. From these challenges, dif-

ferent non-functional requirements were stated that should help improve or prevent issues that may arise and help create user-defined gesture interactions that are not only intuitive to define, but also intuitive to use. More user-centered aspects like intuitiveness and recall of gestures were discussed, while also keeping aspects of the implementation like gesture recognition and end-user authoring paradigms in mind. A set of design principles was proposed with the intention to be general principles for a better implementation of user-defined gesture interaction, but specific examples of solutions for certain problems were also provided for each of the challenges they address.

Lastly, the existing end-user authoring tool called eSPACE was used to provide a use case for implementing user-defined gesture interaction, with use of the earlier proposed design principles. Before starting the implementation, a survey was done where participants were asked about their preferences in terms of some of the user interface design decisions, to assess the design decision we have made and solutions we have come up with, but also to decide between different possibilities we had not decided on yet. After interpreting the results of the survey, the implementation was done according to the results of the survey, but also according to the earlier defined design principles. Each of the principles was discussed in the context of our implementation and the takeaways of this were used to improve our implementation. After the implementation was done, a user study was conducted to get quantitative and qualitative feedback on the usability of the system. The results of this user study were then used to evaluate the system that was implemented and to come up with possible improvements that could be made.

6.1 Conclusion

We were able to achieve our objectives of identifying challenges, proposing solutions and creating design principles to help with the implementation of user-defined gesture interaction into applications, more specifically focused on end-user authoring tools. The proposed design principles helped at making certain design decisions on a more global, conceptual level, but the earlier research also helped at solving certain specific challenges that occurred during the design process.

In terms of the type of gestures we chose to implement For the *incentive* principle, we considered the advantages and disadvantages of implementing gesture interactions, and more specifically user-defined gesture interactions into eSPACE. Considering the advantages and disadvantages also helped

identify challenges in terms of our specific use case. After deciding that the advantages outweigh the disadvantages, the *location of definition* of the gesture interactions was considered. For *consistency* with the current system, the gesture interactions are defined as rules, just like the other interactions in eSPACE. These rules can be edited in the *rules* view and *interaction* view of the tool, but we also decided to allow the user to edit the gesture interactions in the *app* view because this is where the gestures will be used and we already have the Extra-UI with a list of the defined gesture interactions, so we allowed the users to edit this list by adding, removing or editing the interactions.

This Extra-UI contains the list of all defined gesture interactions, and the gesture catalogue, which shows drawings of all the gestures, such that the user knows which gestures exist and how they should be performed. The ideas of the Extra-UI and both the list of defined interactions and the gesture catalogue were conceived in context of the *clarity* principle, which addresses the “invisibility” of gesture interactions, where the user cannot visibly see which gesture interactions are available in an application. Gesture interactions also do not always provide visual feedback when they are performed. For this, we thought of the idea to either show a drawing on the screen of the gesture that was performed for a brief amount of time after it was performed, or to show a snackbar message at the bottom of the screen, which tells the user what gesture was performed and what action was triggered. We chose the latter option.

In terms of the *safety* principle, we took this into consideration when choosing the set of gestures. From the predefined set of gestures that were included in the implementation of the gesture recogniser, we selected a set of gestures that were as distinct as possible, such that the chance of wrong recognition of a gesture is as low as possible. There is still the possibility to add gestures, for example, someone in the user study suggested to add some gestures, like a simple “swipe” gesture, which is definitely possible, but for each gesture added, the recognition accuracy must be considered. Lastly, the *interaction levels* is a concept where we consider on which level the gestures interact with the application. For example, the gestures could be available all throughout the application, only on a certain page, or only on a certain device, or they could only be performed on certain UI elements. For eSPACE, the former two are the same, as there is only a single page in the *app view*, which can have different UIs on different devices. We chose to allow for both gestures that can be performed on the whole page, or targeted on

a certain UI element, but left out the option of having different gestures on different devices, as this would introduce extra complexity for the users and the experiments.

All in all, even if the proposed principles can be improved upon and others can be added, they did show merit at solving challenges that are introduced by implementing user-defined gesture interaction and even showed useful when trying to find solutions for specific problems introduced by a real-world use case of an end-user authoring tool. There seems to be a trade-off between giving the user more freedom to customise their experience in an application and the perceived difficulty of the application, but if we provide the user with sufficient tools to create interactions that are intuitive to them, we can find a balance between the two that seems just right. To answer the question of “How to implement user-defined gesture interaction in an easy-to-use and intuitive manner?”, we can say that building a robust system that is user-centered, which gives the user enough freedom to define the interactions they deem to be intuitive, while also providing the user with enough tools to overcome the weaknesses of gesture interfaces, like the invisibility of gesture interactions, by having visual indications of what gesture was performed, or which gesture interactions are available in the application. Allowing the users quick and easy access to the Extra-UI where they could quickly look up and change, add and remove gesture interactions seemed well appreciated by the users in the user study. Choosing how much freedom to give to the user depends on what the target user is of the application and what the designer of the application is trying to achieve by allowing user-defined gesture interaction, or in other words, their *incentive* for implementing it in the application.

6.2 Future Work

In terms of future work, there is one theme that we avoided, which introduces many opportunities, but also challenges when trying to design an easy-to-use, intuitive and robust system, namely, allowing the user to create custom gestures which they can use in their gesture interactions. With the gesture recogniser we choose for our implementation, we left this option open, as the \$1 Gesture Recognizer allows for easy prototyping of gestures as it uses a defined set of **Unistrokes**, which are sets of points that describe a gesture. We could easily add gestures to this, which is also something that can be done on the website of the implementation we used¹, where there is a canvas

¹<http://depts.washington.edu/accelab/proj/dollar/index.html>

embedded into the page, as depicted in Figure 6.1 which allows users to perform gestures which can be added to the set of defined gestures.

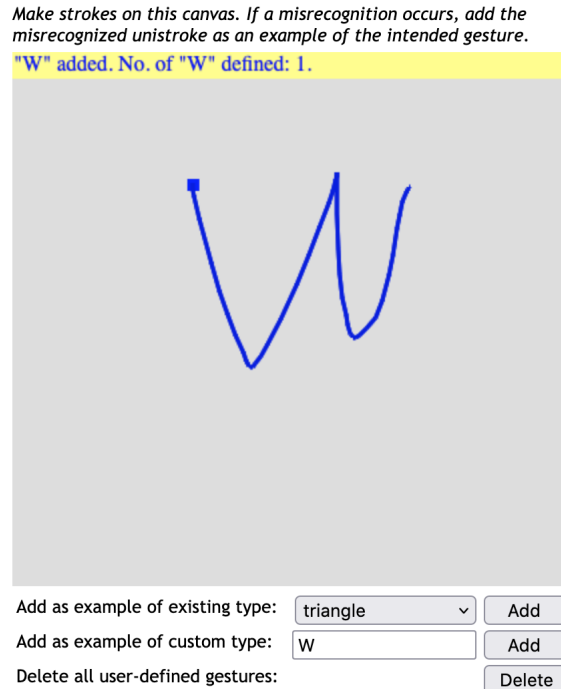


Figure 6.1: A canvas with which custom gestures can be added to the set of defined gesture

With our current implementation, we could allow the user to add, and possibly remove, gestures from the defined set of gestures by drawing it on a canvas and giving it a name, or we could allow them to directly draw the gesture when defining a gesture interaction, without even adding it to the list of defined gestures.

Other possibilities to build on the work we have done are to either implement the improvements that were proposed after doing the evaluation of the implementation, or to implement some of the other ideas that were proposed, but not implemented. For example, showing all defined interaction rules in the Extra-UI is an idea that was explored, and an idea we also asked the participants of the survey about, but which was not implemented. When all interactions are shown, and possibly edited, in the Extra-UI, it could become a long list for certain applications, and the specific implementation would still have to be thought out, but we suggested to group the rules based on the properties of the rules. What we suggested, because the rules are all

“trigger-action” rules, where when a certain trigger is performed, an action gets triggered, was to group the rules either by their trigger or by their action. A majority of people preferred to have the grouping done by the type of trigger, while one person suggested to make this option configurable by the user.

Adding options for the users to change the Extra-UI to their personal preferences in terms of grouping of rules, or rule notation could also be explored. These options could also allow for the snackbar to show or not when a gesture is performed, or any other design decisions which seem more like personal preference than having an objective better or worse option.

We also mentioned the option to allow for different gesture interactions, or even different gestures altogether for each device. There are many directions in which one could go with this, for example, each device could have the same set of gestures, but different gesture interaction rules. Some of the rules could be shared among all the devices, or this idea could be combined with the idea of custom gestures. Another option is to have different gestures for each device, which could be all touch gestures, or they could even be different types of gestures, based on the sensors available in the device. How these gestures could be defined per device is already something we explored in our discussion, but further depth into the topic is certainly possible, and testing it in a practical setting could provide with more interesting insight than our theoretical discussion did.

A

Survey

Demographics

Dear volunteer,

Thank you for participating in this research. We greatly appreciate your interest and cooperation! The questionnaire takes *around 10 minutes* to complete.

In order to protect your opinions, your answers are treated *confidentially* in accordance with the Belgian and European privacy legislation (Cf. AVG or GDPR). All your answers will be processed *anonymously*, so your identity is never revealed.

Contact:

Name: Bram Dewit

Email: brdewit@vub.be

There are a maximum of 13 questions in this survey.

A note on privacy

This survey is anonymous. The record kept of your survey responses does not contain any identifying information about you unless a specific question in the survey has asked for this. If you have responded to a survey that used an identifying token to allow you to access the survey, you can rest assured that the identifying token is not kept with your responses. It is managed in a separate database, and will only be updated to indicate that you

have (or haven't) completed this survey.
There is no way of matching
identification tokens with survey
responses in this survey.

What is your year of birth?

Gender?

- Male
- Female
- Other

Highest level of school you have completed or highest degree you have received?

- Less than high school degree
- High school degree or equivalent
- Bachelor's degree
- Master's degree
- PhD degree

During your studies, did you get in contact with programming and/or user interface design (computer science-related courses)?

- Yes
- No

How much computer science-related activities did you follow?

- Not much (1-2 courses)
- A few (3-4 courses)
- Many (I am a computer scientist)

In which country were you born?

Survey

First of all, **thank you** for taking the time to fill in this survey!

For my master's thesis I investigate how user-defined **gesture interaction** can be intuitively integrated into **authoring tools**, and what the **advantages** and **challenges** are of doing so.

End-user authoring tools are tools which allow **users** to create applications **without** requiring any **programming knowledge**. These users can **create user interfaces** (UIs) and define their behaviour, solely using the tool. The behaviour my research is focused on, are the gesture interactions. These can be **touch gestures** (like swipes), or any other type of gestures (like mid-air hand gestures).

Unlike UI elements (such as buttons and sliders), gesture interactions are **not visible**, which can pose **challenges** in terms of **clarity**, like knowing which gestures are defined, or knowing when a gesture gets triggered. In this survey, we will show you **different versions** of a user interface which purpose is to **show the available interactions** of a specific application and which allows users to edit these interactions.

The survey will **query your preferences** in terms of the **structure** and **layout** of this interface.

The **behaviour** of the application is defined by certain "**interaction rules**". The special

user interface to view and/or edit these interaction rules will be opened by clicking a button at the top of the application.

Below you can see a mockup of the layout of this UI, in this example 2 interaction rules are defined. One where a "Triangle" gesture is performed on the page and turns a light on, while performing the "Circle" gesture on the page will turn the light off.

*note that all images shown are drawn mockups and the actual UI looks different

Interaction Rules		Edit Rules				
IF gesture:	<input type="text" value="Triangle"/>	on:	<input type="text" value="Page"/>	THEN	<input type="text" value="Light On"/>	✖
IF gesture:	<input type="text" value="Circle"/>	on:	<input type="text" value="Page"/>	THEN	<input type="text" value="Light Off"/>	✖
<input type="button" value="Add"/>						
						<input type="button" value="Close"/> <input type="button" value="Save"/>

We want to allow users to view and **edit** interaction rules. As potential user of this application, would you rather have a **single screen** where the rules can be edited (and consequently viewed)? Or would you prefer to have a **separate screen**, one for viewing and another one for editing these rules?

- Single rule editing screen
- Separate rule viewing & rule editing screen

Would you prefer to see both in a **single pop-up screen with 2 tabs** (one for viewing, one for editing), or would you prefer **2 different buttons** which open up the **2 different pop-up screens**?

- Single button opening tabbed pop-up screen
- Two buttons opening different screens

The interaction rules follow a so-called "**trigger-action**" pattern, which means a certain event (like a button click, or a gesture that is performed) *triggers* a certain *action* (such as "Light on") by the application.

Currently **user-initiated** triggers and **contextual** triggers are defined. The first type can be a button press or gesture, and the second one can be a certain time of day or the location of the device.

To show a more **structured overview** of the list of rules, they could be divided based on these types.

The "user-initiated" interaction type could be divided further into categories like "UI interaction", "gesture interaction", and "device interaction".

Below graphic shows an example of the rules shown grouped by the type of trigger they use:

Interaction Rules	Edit Rules
<p>Gesture Interactions:</p> <ul style="list-style-type: none">- IF gesture: Triangle on Page THEN Turn lights on- IF gesture: Circle on Page THEN Turn lights off- IF ... <p>Contextual Interactions:</p> <ul style="list-style-type: none">- IF Time = 17:30 THEN Thermostat: Turn on heating- IF ... <p>Device Interactions:</p> <ul style="list-style-type: none">- IF Bram's Phone wifi turned on THEN Synchronise images with Bram's Laptop- IF ... <p style="text-align: center;">• • •</p>	

Would you like to have these **categories** based on the **type of trigger** for each rule, or would you prefer **other categories**?

- No categories (just 1 list of all the rules)
- Categories based on the type of trigger
- Categories based on the type of action
- Other:


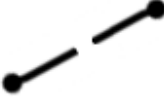







Multiple notations of the rules are possible. Would you prefer to see the rules written as "IF trigger **THEN** action" (current notation in the tool), with an arrow notation like "trigger ➔ action", or do you have another suggestion?

- IF trigger **THEN** action
- trigger ➔ action
- Other:


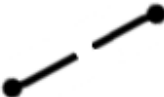







My research is focused on how to **integrate gesture interactions**, which introduces multiple challenges.

One issue is that the user might not know **all possible gestures** that can be performed, or what they would look like, one possibility is to add a "**gesture catalogue**" showing the name and a visualisation of each of the gestures.

Example layout of the gesture catalogue:

Edit Interaction Rules	Gesture Catalogue	
 <p data-bbox="272 495 442 533">Right Swipe</p>	 <p data-bbox="632 495 713 533">Pinch</p>	 <p data-bbox="922 486 1072 524">Left Swipe</p>
 <p data-bbox="304 808 421 846">Triangle</p>	 <p data-bbox="635 808 719 846">Circle</p>	 <p data-bbox="983 808 1010 846">X</p>
 <p data-bbox="344 1115 363 1153">V</p>	 <p data-bbox="639 1115 724 1153">Caret</p>	 <p data-bbox="954 1115 1023 1153">Loop</p>

Example layout of the gesture catalogue:

Interaction Rules	Edit Rules	Gesture Catalogue
 Right Swipe	 Pinch	 Left Swipe
 Triangle	 Circle	 X
 V	 Caret	 Loop

Would you want a **gesture catalogue**, and if so, **where** would you add it?

- I don't want a gesture catalogue
- Add it in a tab on the "view rules" screen
- Add it in a tab on the "edit rules" screen
- Add it to both screens
- Add a third button to the application opening a separate screen with the catalogue

Would you want a **gesture catalogue**, and if so, **where** would you add it?

- I don't want a gesture catalogue
- Add it in another tab
- Add a second button to the application opening a separate screen with the catalogue

For the last question, we would like to know what **type of interface** you would prefer to **edit** the **gestures**. One possibility is, like in the previously shown designs, where the gesture is selected using a **dropdown menu**.

Another option would be to use a button that opens a pop-up window with the **gesture catalogue**, where the user can click on the gesture they want to use.

Lastly, we could add **icons** for each gesture to the **dropdown**, such that the user has both a visual and textual representation of the icon. Assuming we have a gesture catalogue, the user could still look up the exact shape of the gesture in the catalogue.

Which approach do you prefer?

- Dropdown with only text
- Dropdown with icon and text
- Pop-up window with gesture catalogue

© Vrije Universiteit Brussel 2020

Powered by Qualtrics

Bibliography

- [1] Roland Aigner, Daniel Wigdor, Hrvoje Benko, Michael Haller, David Lindbauer, Alexandra Ion, Shengdong Zhao, and Jeffrey Tzu Kwan Valino Koh. Understanding Mid-Air Hand Gestures: A Study of Human Preferences in Usage of Gesture Types for HCI. *Microsoft Research TechReport MSR-TR-2012-111*, 2:30, 2012.
- [2] Jonathan Alon, Vassilis Athitsos, Quan Yuan, and Stan Sclaroff. A Unified Framework for Gesture Recognition and Spatiotemporal Gesture Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(9):1685–1699, 2009.
- [3] Anbarasan and Jeannie Su Ann Lee. Speech and Gestures for Smart-Home Control and Interaction for Older Adults. In Susanne Boll, Ramesh C. Jain, Noel E. O’Connor, Troy McDaniel, and Jochen Meyer, editors, *Proceedings of the 3rd International Workshop on Multimedia for Personal Health and Health Care, HealthMedia@MM 2018, Seoul, Republic of Korea, October 22, 2018*, pages 49–57. ACM, 2018.
- [4] Andy Cockburn, Carl Gutwin, Joey Scarr, and Sylvain Malacria. Supporting Novice to Expert Transitions in User Interfaces. *ACM Comput. Surv.*, 47(2):31:1–31:36, 2014.
- [5] David Garlan, Robert Allen, and John Ockerbloom. Architectural Mismatch: Why Reuse Is So Hard. *IEEE Softw.*, 12(6):17–26, 1995.
- [6] Xiao-Li Guo and Ting-Ting Yang. Gesture recognition based on HMM-FNN model using a Kinect. *J. Multimodal User Interfaces*, 11(1):1–7, 2017.
- [7] Mikkel R. Jakobsen, Yvonne Jansen, Sebastian Boring, and Kasper Hornbæk. Should I Stay or Should I Go? Selecting Between Touch and Mid-Air Gestures for Large-Display Interaction. In Julio Abascal, Simone Barbosa, Mirko Fetter, Tom Gross, Philippe Palanque, and

- Marco Winckler, editors, *Human-Computer Interaction – INTERACT 2015*, pages 455–473, Cham, 2015. Springer International Publishing.
- [8] Daehwan Kim and Daijin Kim. An Intelligent Smart Home Control Using Body Gestures. In *2006 International Conference on Hybrid Information Technology*, volume 2, pages 439–446, 2006.
- [9] Minseok Kim, Sung Ho Choi, Kyeong-Beom Park, and Jae Yeol Lee. User Interactions for Augmented Reality SmartGlasses: A Comparative Evaluation of VisualContexts and Interaction Gestures. *Applied Sciences*, 9(15):3171, 2019.
- [10] Kenrick Kin, Björn Hartmann, Tony DeRose, and Maneesh Agrawala. Proton: Multitouch Gestures as Regular Expressions. In Joseph A. Konstan, Ed H. Chi, and Kristina Höök, editors, *CHI Conference on Human Factors in Computing Systems, CHI '12, Austin, TX, USA - May 05 - 10, 2012*, pages 2885–2894. ACM, 2012.
- [11] Vassilis Kostakos and Mirco Musolesi. Avoiding pitfalls when using machine learning in HCI studies. *Interactions*, 24(4):34–37, 2017.
- [12] David M. Lane, H. Albert Napier, S. Camille Peres, and Aniko Sandor. Hidden Costs of Graphical User Interfaces: Failure to Make the Transition from Menus and Icon Toolbars to Keyboard Shortcuts. *Int. J. Hum. Comput. Interact.*, 18(2):133–144, 2005.
- [13] Jaime Lien, Nicholas Gillian, Mustafa Emre Karagozler, Patrick Amihoud, Carsten Schwesig, Erik Olson, Hakim Raja, and Ivan Poupyrev. Soli: Ubiquitous Gesture Sensing with Millimeter Wave Radar. *ACM Trans. Graph.*, 35(4):142:1–142:19, 2016.
- [14] Wanhong Lin, Lear Du, Carisa Harris-Adamson, Alan Barr, and David Rempel. Design of Hand Gestures for Manipulating Objects in Virtual Reality. In Masaaki Kurosu, editor, *Human-Computer Interaction. User Interface Design, Development and Multimodality - 19th International Conference, HCI International 2017, Vancouver, BC, Canada, July 9-14, 2017, Proceedings, Part I*, volume 10271 of *Lecture Notes in Computer Science*, pages 584–592. Springer, 2017.
- [15] Allan Christian Long Jr., James A. Landay, and Lawrence A. Rowe. Implications for a Gesture Design Tool. In Marian G. Williams and Mark W. Altom, editors, *Proceeding of the CHI '99 Conference on Human Factors in Computing Systems: The CHI is the Limit, Pittsburgh, PA, USA, May 15-20, 1999*, pages 40–47. ACM, 1999.

-
- [16] Hao Lü and Yang Li. Gesture Coder: A Tool for Programming Multi-Touch Gestures by Demonstration. In Joseph A. Konstan, Ed H. Chi, and Kristina Höök, editors, *CHI Conference on Human Factors in Computing Systems, CHI '12, Austin, TX, USA - May 05 - 10, 2012*, pages 2875–2884. ACM, 2012.
- [17] Hao Lü and Yang Li. Gesture Studio: Authoring Multi-Touch Interactions through Demonstration and Declaration. In Wendy E. Mackay, Stephen A. Brewster, and Susanne Bødker, editors, *2013 ACM SIGCHI Conference on Human Factors in Computing Systems, CHI '13, Paris, France, April 27 - May 2, 2013*, pages 257–266. ACM, 2013.
- [18] Nathan Magrofuoco, Paolo Roselli, and Jean Vanderdonckt. Two-dimensional Stroke Gesture Recognition: A Survey. *ACM Comput. Surv.*, 54(7):155:1–155:36, 2022.
- [19] Jérémie Melchior, Jean Vanderdonckt, and Peter Van Roy. A Comparative Evaluation of User Preferences for Extra-User Interfaces. *Int. J. Hum. Comput. Interact.*, 28(11):760–767, 2012.
- [20] Martin Mihajlov, Effie Lai-Chong Law, and Mark Springett. Intuitive Learnability of Touch Gestures for Technology-Naïve Older Adults. *Interact. Comput.*, 27(3):344–356, 2015.
- [21] Donald A. Norman and Jakob Nielsen. Gestural Interfaces: A Step Backward In Usability. *Interactions*, 17(5):46–49, 2010.
- [22] Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. Design-Scape: Design with Interactive Layout Suggestions. In Bo Begole, Jinwoo Kim, Kori Inkpen, and Woontack Woo, editors, *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015*, pages 1221–1224. ACM, 2015.
- [23] Uran Oh and Leah Findlater. The Challenges and Potential of End-User Gesture Customization. In Wendy E. Mackay, Stephen A. Brewster, and Susanne Bødker, editors, *2013 ACM SIGCHI Conference on Human Factors in Computing Systems, CHI '13, Paris, France, April 27 - May 2, 2013*, pages 1129–1138. ACM, 2013.
- [24] Thomas Plötz. Applying Machine Learning for Sensor Data Analysis in Interactive Systems: Common Pitfalls of Pragmatic Use and Ways to Avoid Them. *ACM Comput. Surv.*, 54(6):134:1–134:25, 2021.

- [25] Seema Rawat, Somya Vats, and Praveen Kumar. Evaluating and exploring the Myo Armband. In *2016 International Conference System Modeling & Advancement in Research Trends (SMART)*, pages 115–120, 2016.
- [26] Audrey Sanctorum. *eSPACE: Conceptual Foundations for End-User Authoring of Cross-Device and IoT Applications*. PhD thesis, Vrije Universiteit Brussel, 2020.
- [27] Barry Schwartz. *The Paradox of Choice: Why More Is Less*. Harper Perennial, January 2005.
- [28] Davy Vanacken, Alexandre Demeure, Kris Luyten, and Karin Coninx. Ghosts in the Interface: Meta-user Interface Visualizations as Guides for Multi-touch Interaction. In *Third IEEE International Workshop on Tabletops and Interactive Surfaces (Tabletop 2008), October 1-3 2008, Amsterdam, The Netherlands*, pages 81–84. IEEE Computer Society, 2008.
- [29] Robert Walter, Gilles Bailly, and Jörg Müller. StrikeAPose: Revealing Mid-Air Gestures on Public Displays. In Wendy E. Mackay, Stephen A. Brewster, and Susanne Bødker, editors, *2013 ACM SIGCHI Conference on Human Factors in Computing Systems, CHI '13, Paris, France, April 27 - May 2, 2013*, pages 841–850. ACM, 2013.
- [30] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. Gestures without Libraries, Toolkits or Training: A \$1 Recognizer for User Interface Prototypes. In Chia Shen, Robert J. K. Jacob, and Ravin Balakrishnan, editors, *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology, Newport, Rhode Island, USA, October 7-10, 2007*, pages 159–168. ACM, 2007.
- [31] Wendy Yee. Potential Limitations of Multi-touch Gesture Vocabulary: Differentiation, Adoption, Fatigue. In Julie A. Jacko, editor, *Human-Computer Interaction. Novel Interaction Methods and Techniques, 13th International Conference, HCI International 2009, San Diego, CA, USA, July 19-24, 2009, Proceedings, Part II*, volume 5611 of *Lecture Notes in Computer Science*, pages 291–300. Springer, 2009.