2013-01-25

Designing Tabletop and Surface Applications Using Interactive Prototypes

de Souza Alcantara, Tulio

UNIVERSITY OF CALGARY

Designing Tabletop and Surface Applications Using Interactive Prototypes

by

Túlio de Souza Alcantara

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

JANUARY, 2013

ABSTRACT

Innovative possibilities of interacting with touch-based devices come with the cost of having interaction designers to determine if users consider the interactions natural and easy to use. It is safe to assume that user-centered design helps creating applications that better fit user requirements and one of its steps is prototyping. However, interaction designers sometimes skip the prototyping phase due to time or economic constraints or for a lack of proper tools. In order to help designers of touch-based applications this research presents an approach that allows designers to create their own application-specific hand, finger or tag-based gestures and evaluate these gestures using sketch-based prototypes for touch-based applications without requiring any textual programming. The development of a sketch based prototyping tool followed a user centered design approach with requirements provided by user experience designers from industry. The whole application suite was evaluated to determine its fitness for helping designers of touch-based applications. The evaluation process was conducted in three steps: first, a study with software developers determined the efficacy of the tool to create custom gestures for performing tasks in sketch based prototypes. A follow up evaluation investigated the performance of the tool as a prototyping tool for tangible applications. Finally this thesis reports two cases where this solution helped in the design of real touch-based applications.

ACKNOWLEDGMENTS

In a two years project, there are several people that help you a lot in ways that it might even take a while for one to realize. Of course, there are the obvious ones that are there to help and guide you: my supervisor Dr. Frank Maurer for all the support, guidance and for believing that I could get the job done and done well; Dr. Jennifer Ferreira for the innumerous discussions and advices; to Dr. Jörg Denzinger, for the discussions and for challenging me; and for all the professors that I had the pleasure to work with, as a student or as a teacher assistant.

I also want to thank all my friends from the ASE lab that have been a part of my life in these last two years. I won't get emotional with you guys here, because that is not how we roll, but I want to thank you all for the support, the jokes (quite often inappropriate) and just for being there.

I also want to thank the support from my friends: the new ones that I made here and the ones from Brazil. Because sometimes, the only thing that will help you fix that line of code that is not working or to write that paragraph that doesn't feel right, is to go out and party. Well, sometimes studying also helps, but somehow, it is not quite the same.

I want to express my eternal gratitude to Larissa, for being supportive, understanding and for making hard days happier and cheerful.

Finalmente, eu quero agradecer à minha familia: painho, mainha e Val. Como eu sempre disse, vocês sãoo o meu porto seguro, e tudo que eu faço e conquisto tem muito da forca que voces depositam em mim. Aonde quer que eu esteja, sempre levarei voces comigo.

UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled “Designing Tabletop and Surface Applications Using Interactive Prototypes” submitted byTulio de Souza Alcantara in partial fulfilment of the requirements for the degree of Master of Science

Supervisor, Dr. Frank O Maurer
Department of Computer Science

Dr. Mario Costa Sousa
Department of Computer Science

External Examiner Dr. Michele Jacobsen
Educational Research

January, 21st 2013

Co-Author Permission

University of Calgary

2500 University Drive NW

Calgary, Alberta

T2N 1N4

January, 24th 2013

I, Frank Maurer, grant Tulio de Souza Alcantara full permission to use the content of the following co-authored publications in his M.Sc. dissertation and to have this work microfilmed.

de Souza Alcantara, T.; Denzinger, J.; Ferreira, J.; Maurer, F.; , "Learning gestures for interacting with low-fidelity prototypes," Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2012 First International Workshop on , vol., no., pp.32-36, 5-5 June 2012

Co-Author Permission

University of Calgary

2500 University Drive NW

Calgary, Alberta

T2N 1N4

January, 24th 2013

I, Jennifer Ferreira, grant Tulio de Souza Alcantara full permission to use the content of the following co-authored publications in his M.Sc. dissertation and to have this work microfilmed.

de Souza Alcantara, T.; Denzinger, J.; Ferreira, J.; Maurer, F.; , "Learning gestures for interacting with low-fidelity prototypes," Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2012 First International Workshop on , vol., no., pp.32-36, 5-5 June 2012

Co-Author Permission

University of Calgary

2500 University Drive NW

Calgary, Alberta

T2N 1N4

January, 24th 2013

I, Jörg Denzinger, grant Tulio de Souza Alcantara full permission to use the content of the following co-authored publications in his M.Sc. dissertation and to have this work microfilmed.

> de Souza Alcantara, T.; Denzinger, J.; Ferreira, J.; Maurer, F.; , "Learning gestures for interacting with low-fidelity prototypes," Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2012 First International Workshop on , vol., no., pp.32-36, 5-5 June 2012

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABREVIATIONS

WIMP	Windows, Icons, Menus, Pointers
HCI	Human Computer Interaction
UI	User Interface
ITS	Interactive tabletop and surface
IGT	Intelligent Gesture Toolkit
GDL	Gesture Definition Language
BLOB	Binary Large Objects

LIST OF TABLES

PUBLICATIONS

de Souza Alcantara, T.; Denzinger, J.; Ferreira, J.; Maurer, F.; , "Learning gestures for interacting with low-fidelity prototypes," Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2012 First International Workshop on , vol., no., pp.32-36, 5-5 June 2012

CHAPTER ONE : INTRODUCTION

"We must design our technologies for the way people actually behave, not the way we would like them to behave" [1 p. 12]

Designing *Windows, Icons, Menus and Pointers* (WIMP) based applications is a well-known challenge. This challenge becomes even bigger to design for touch-based devices and gesture based applications [3, 5, 6, 11, 12, 34 and 35]. The increasing popularity of multi-touch tabletop and surface computing opens up new possibilities for interaction paradigms, which allows designers to design applications that can be interacted with in new and different ways, through gesture-based and touch-based interactions that can improve or hamper the user experience [1, 2 and 3]. *Interactive Tabletops and Surfaces* (ITS) are highly visual systems, which are usually controlled by touches and gestures performed on the device, enabling users to directly interact with the application using their hands or tangible objects. For ITS applications, a preferable user interface integrates gesture-based interactions into the applications [6]. Frameworks such as *Windows Presentation Foundation* (WPF) [7] provide a set of pre-defined gestures that application developers can use easily [8 and 9]. However, the literature shows many examples of gestures that are not available *out of the box* [6 and 10]. When creating gestures for interacting with ITS applications, interaction designers have to determine if users consider them natural, understandable and easy to use [6].

In the context of ITS applications, designers can explore innovative ways for users to interact with their applications; this design might drastically hamper the user experience if *Human Computer Interaction* (HCI) principles are not taken in consideration [2]. What is necessary is a way to help designers follow HCI principles not only on the design of the interface of ITS applications but also the interactions. Previous research on gesture-based interaction has shown problems with the design of gestures, the meaning of touch and gestures and how context influences them [3, 5, 6, 11, 12, 34 and 35]. In the gesture design scenario, there are two main challenges:

- the effort, time and technical expertise required to create gestures [13, 14 and 15];
- the design of gestures that are suitable for specific tasks, context and users [5 and 12].

Research in the current state of design of multi-touch applications shows a lack of processes and tools to support the design of these applications [3, 5 and 16]. The authors of these studies bring up the need to allow designers to follow up methods to improve the design of multi-touch applications, such as user-centered design [35].

Having users involved early in the process through iterative prototypes has being widely researched and the advantages of sketching and prototyping to improve the design of applications has been proved successful [4, 17, 18, 19, 20, 21 and 22]. Especially in the novel scenario of gesture and tangible based applications, where Norman and Nielsen [2] argued that gestures might be harmful for usability designers need to evaluate if the gestures are improving usability. As shown by Moggride [4], Krippendorff [23] and Buxton [24] sketching has shown to be a valuable aid to designers in order to validate ideas with users in early stages of the design. In any activity of design, sketching has been proved to be a crucial part of it and several contributions [4, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 31, 37, 39 and 42] defend the importance and the benefits of sketching and prototyping to improve design ideas by failing early, often and then, learning from mistakes. While these authors defend the use of paper as a medium to transmit ideas, sketching the dynamic of applications lacks tool support [24] especially in applications that use innovative interactions, mainly ITS applications.

Memmel *et al.* [25] studied how prototyping can elicit requirements. They recommend the use of abstract prototypes, as filling in details too early might lead to premature decisions, leading to wasted effort and time spent on these details. Abstract prototypes help designers get important aspects of the content and organization of the user interface while deferring details about how the final application will look like and operate [26]. These studies motivated this research by showing the importance of prototyping in a UCD process which ultimately can help designers fit the workflow in agile iterations.

Prototypes allow designers to evaluate the output of a system, while the input is assumed to be obvious; they allow designers to evaluate what users want to do, while the interactions and how users want to do certain tasks is not a trivial task [22]. If the interaction input is more complex, paper prototypes are not sufficient [17]. Based on this limitation, the motivation to this thesis is to make developing of usable and gesture-based applications easier and better fitting user's needs. For this issue, a desired solution has to:

- make it easy to design gestures, respecting the time and cost constraints of prototyping;
- make it easy to evaluate if these gestures are usable.

1.1 Research Goals

This research focuses on identifying the challenges in designing the interface and interaction of multi-touch and tangible applications. The questions the author wants to answer are:

- new technologies allow new and unconventional ways of interaction. How can this solution serve as a basis to help basic HCI principles such as user-centered design to be followed in designing new interaction ideas?
- How does a prototyping tool that can evaluate custom gestures affect the design of touch and tag-based applications?
- What are the benefits of allowing designers to create custom gestures?

The overall goal of this research is to provide designers with a tool that helps them deliver highly useable ITS applications (multi-touch, gestures and tangibles). This goal consists of the following sub-goals:

- helping designers create custom gestures with no programming effort;
- support quick idea generation through sketching;
- support usability studies through prototyping evaluation.

The chosen approach is to provide designers with a sketch based tool for evaluating ITS interfaces and interactions in the context of the tasks in the application.

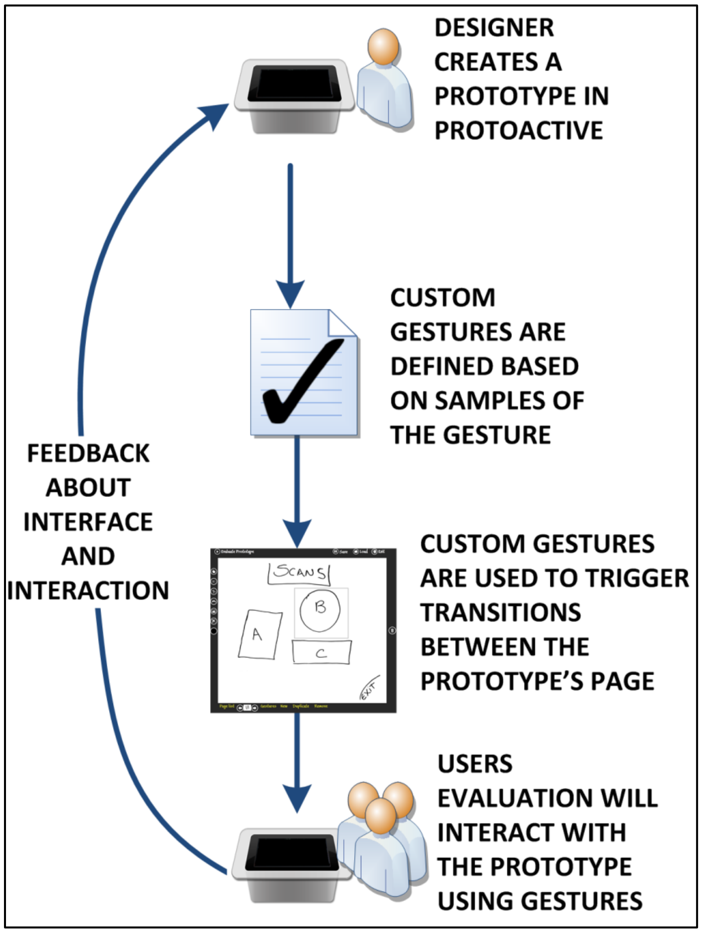
1.2 Contributions

This research has two main contributions: first, ProtoActive which is composed of:

- a storyboarding sketching tool, designed to draft prototypes for ITS applications;
- an integrated gesture learning tool – *Intelligent Gesture Toolkit* (IGT) that uses a specialized machine learning (anti-unification) method to create gesture definitions based on samples of gestures;

- a tool that emulates ITS applications and allows designers to gather feedback on their usability.

Second, this research provides a study that evaluates the benefits and limitations of ProtoActive. An overview of ProtoActive functionality is shown in Figure 1. It explains the cycle proposed in this research, where a designer creates a prototype of an ITS application and evaluates the interactions in it, having through this, early user feedback about the design of the interface and interactions of the application. The whole cycle shown in Figure 1 can be done without any programming effort or any of the implications of it e.g.: setting up work station or managing source code.


DESIGNER CREATES A PROTOTYPE IN PROTOACTIVE
CUSTOM GESTURES ARE DEFINED BASED ON SAMPLES OF THE GESTURE
CUSTOM GESTURES ARE USED TO TRIGGER TRANSITIONS BETWEEN THE PROTOTYPE'S PAGE
USERS EVALUATION WILL INTERACT WITH THE PROTOTYPE USING GESTURES
FEEDBACK ABOUT INTERFACE AND INTERACTION
SCANS
A
B
C
EXIT
Figure 1 ProtoActive evaluation cycle

1.3 Thesis Structure

The following chapter discusses background information necessary to better understand this thesis. This is followed by related work of this thesis, showing studies that relate to this research. Chapter Four explains ProtoActive in detail, showing the user centered design approach used to design it and how it integrates with IGT. Chapter Five explains IGT, the gesture definition tool and its contributions regarding the anti-unification approach used to learn custom gestures and the capabilities of the gesture recognizer framework used, including it capabilities to detect hand postures and fiduciary markers. Chapter Six shows the evaluation of our solution, how it was conducted and how the results found answer our research questions. Finally Chapter Seven concludes this thesis, showing the contributions of our findings and how this thesis can be improved as a future work in the area.

CHAPTER TWO : BACKGROUND

In order to facilitate the understanding of this thesis, this chapter will explain the application domain of the solution proposed here. This will be followed by a presentation based in the current research about the taxonomy of prototyping and how it relates to sketching in the designing domain. Finally, this chapter provides a definition for gestures and tangibles that will be used in this thesis.

2.1 Application domain and technical aspects

ProtoActive is a prototyping tool designed to help designers to create and evaluate *Interactive Tabletop and Surface* (ITS) applications and its interactions. For understanding purpose these interactions will be referenced as gestures and are described with further detail in section 2.2. ProtoActive is currently supported in the following devices:

- Microsoft Surface 1 [27];
- Microsoft Surface 2 (PixelSense) [28];
- Tabletops, tablets or desktop computers with touch enabled monitors, running Microsoft Windows 7 or Microsoft Windows 8.

ProtoActive is a *Windows Presentation Foundation* (WPF) [7] application. It has two versions: one using the Surface SDK 1 and another one using Surface SDK 2. The differences between the two versions are:

- The version for Microsoft Surface 1 uses Microsoft Surface SDK 1 visual controls and the class that manipulates the touch points uses the Microsoft Surface 1 contact points. This version can only be used in a Microsoft Surface 1 device. The sketching canvas on ProtoActive allows multi-touch drawing.

- The version for Microsoft Surface 2 uses WPF standard controls and Microsoft Surface SDK 2 controls. In the Microsoft Surface SDK 2, the touch points from standard WPF applications were unified with Microsoft Surface 2 touch points. This allowed the version for Microsoft Surface 2 to be used in any device running Microsoft Surface 2 Runtime. Due to performance issues, the sketching canvas in ProtoActive for Microsoft Surface 2 had to use a canvas from standard WPF as using the multi-touch canvas from Microsoft Surface SDK 2 showed performance issues for multi-touch. This makes that the drawing canvas for ProtoActive in Microsoft Surface 2 only working with single touch.

ProtoActive gesture learning and recognition mechanisms use a modified version of *Gesture Toolkit* and its proprietary language *Gesture Definition Language* (GDL) using MGrammar [4]. The details of the modifications to *Gesture Toolkit* and to GDL are explained in section 5.3.

2.2 Gestures

As defined by Mitra and Acharya in 2007, gestures can be static or dynamic [29]. In static gestures, a user assumes a certain pose using hands and fingers, while a dynamic gesture can be a stroke or set of strokes on the touch surface. Some gestures combine both, dynamic and static elements. Gestures defined in IGT, differently from stroke based gesture recognizes such as $N [65] and RATA [15], contain information about order, direction and time, allowing IGT to recognize both types of gestures (static and dynamic). ProtoActive contains a gesture recorder (IGT) which allows the designer to create custom gestures and use them during the evaluation of the prototype. In this thesis, gestures are considered as the following interactions on an ITS device:

- single or multi-touch 2D touches;

- hand postures;
- detection of fiduciary markers;
- concurrent interactions (multiple fingers, hands and fiduciary markers).

The gestures defined in IGT are available to be used in prototypes created in ProtoActive, allowing designers to evaluate the interaction of the prototypes through custom gestures. ProtoActive considers the touch based interactions that occur on the surface of the devices, it does not include interactions above tabletop, interactions in front a display (only when it touches), and multi-surface interactions. The detection of fiduciary markers allows the definition of a subset of interactions: tangibles interactions.

2.2.1 Tangibles

By allowing designers to define gestures that detect fiduciary markers, ProtoActive allows designers to prototype the detection of physical tangibles. By attaching a fiduciary tag in a physical object, a designer can use it as a tangible in ProtoActive and have users evaluate them. Figure 27 and Figure 41 illustrate tangibles that can be detected on ProtoActive by their fiduciary markers. Any physical object with a flat surface where a fiduciary mark tag can be attached can be used in ProtoActive.

2.3 Sketching and prototyping

Having more sophisticated and interactive sketches allows designers to take advantages of having users involved and providing feedback about the interactions in ITS applications. Memmel *et al.* [25], propose the iterative use of low-fidelity prototypes in order to validate steps of design and development, resulting in a more iterative and agile process. Further in the design process, sketches can become more sophisticated and goal oriented, thus the time spent onto them changes, also changing the expectations regarding them. This distinction defines the sketches as prototypes [24 p.139].

Buxton [24 p.139] makes a distinction between sketches and prototypes as having different purposes due to the difference between the time spent on them (see Figure 2); even though both are tools that can be used in early stages of the design, sketches are earlier drafts whereas prototypes are created later in the design process when ideas are starting to converge. By using prototypes, not only design ideas but also requirements of software can be brought to attention and properly addressed in early stages of the development. The advantage of prototyping is that it allows designers to experiment and invent [30]. Interactive prototypes then help "*interaction designers to define user interfaces, and evaluate usability issues in early stages of design*" [31]. While designing ITS applications, interactions play an important part in the design of interfaces and can drastically hamper interfaces by having complicated or non-intuitive gestures for interacting with them.

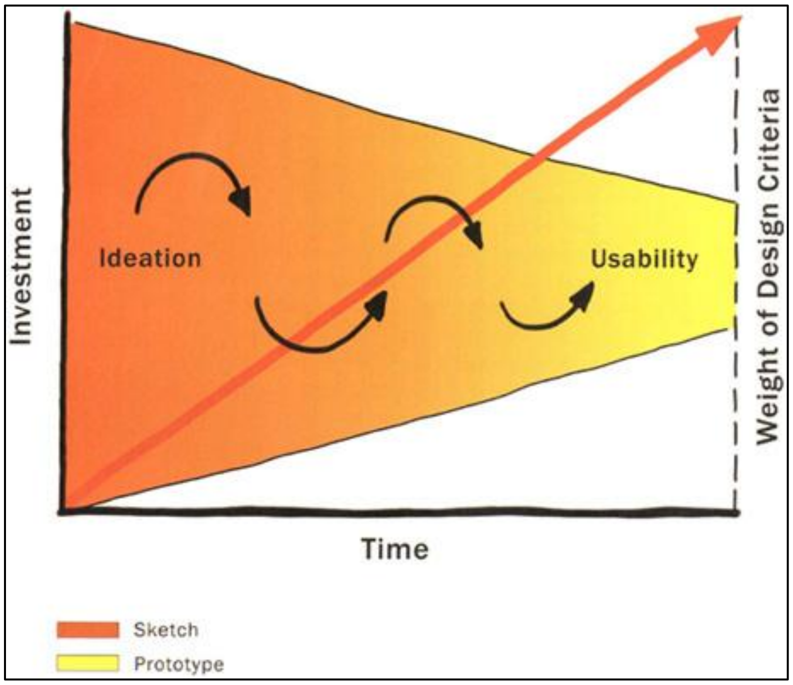


Figure 2 - Sketch x Prototype. Extracted from [24 p.138]

Rudd *et al.* [32] suggest advantages and disadvantages of low-fidelity and high-fidelity prototyping, as shown in Table 1. Considering the advantages and disadvantages in Table 1, the solution proposed in this thesis aims to take the advantages of low-fidelity prototypes and it proposes to address the disadvantages.

Table 1 Low- & High-Fidelity Prototyping, based on [32], extracted from [25]

Type	**Advantages**	**Disadvantages**
Low-Fidelity	less time & lower cost evaluate multiple design concepts communication device address screen layout issues	limited usefulness for usability tests navigational and flow limitations facilitator-driven poor specification
High-Fidelity	partial/complete functionality interactive user-driven clearly defines navigational scheme use for exploration and test marketing & sales tool	time-consuming to create inefficient for proof-of-concept designs blinds users to major representational flaws management may think it is real

The disadvantages that motivated this research are drawbacks in low-fidelity prototyping that could be addressed and some of the features from high-fidelity prototyping that could be incorporated. This research aims in providing a prototyping tool that incorporates interactivity in the level of high-fidelity prototypes allowing usability tests based on interaction but having the low effort cost of low-fidelity prototypes allowing the evaluation of multiple design and interaction concepts. Rudd *et al.* [32] also discuss the flow limitations of low-fidelity prototypes as due to interactivity by the user being somewhat restricted. In low-fidelity prototypes, the user is dependent on the facilitator to respond to the commands of the user to turn pages on the prototype or advance screens to simulate the flow of the application. This issue was addressed by providing a set of facilities that allow designers to create the simulations of different flows with low effort (see section 4.4).

The previous categorization of prototypes in a dichotomy between low-fidelity and high-fidelity is not always valid due to the complexity of some prototypes and their purpose. The following section shows what has been researched to improve a formalization of prototypes categories.

2.4 Types of prototypes

An early study in 1984 by Zelkowitz [33] visualizes the different types and purposes of prototypes as branches in the vertical life of software cycle. An example can be seen in Figure 3.

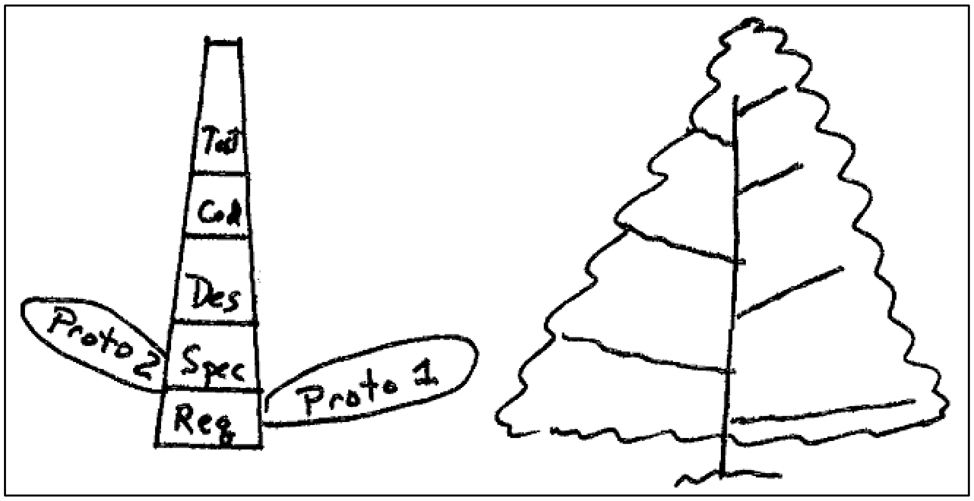


Figure 3 Tree visualization of a prototype in a software life cycle. Extracted from [33]

The study is limited to a certain variety of trees, thus limiting the validity of the study. This study shows that the concern about a taxonomy to categorize prototypes is not a recent problem. Nevertheless, terms need to be coined in order to facilitate communication, so prototypes are commonly classified in a fidelity dichotomy that can be low-fidelity or high-fidelity. Low-fidelity prototypes are considered ostensibly low-cost methods that often are presented as sketches on paper. High-fidelity prototypes have a higher cost and are often used in late stages of development, with a presentation closer to the final product. With the popularity of prototypes, this dichotomy is becoming insufficient to describe the variety of prototypes being used today [20]. In order to avoid misclassifying the prototyping approach used in this thesis, we will first present the current state of research in formalizing the types of prototypes. This will be followed by an explanation about which type of prototypes can be covered with ProtoActive on the light of the presented categories. This should not be mistaken by classifying the prototyping tools as done by Van den Bergh *et al.* [31]. The following studies classify the type of prototype and will be used to classify ProtoActive in terms of the kind of prototypes that can be created with it.

2.4.1 Mixed fidelity

McCurdy *et al.* [20] propose a *mixed-fidelity* classification that differs from *mid-fidelity*. In *mid-fidelity*, prototypes are neither low nor high-fidelity while in *mixed-fidelity* prototypes have aspects of low and high-fidelity. They present a framework for characterizing prototypes along five dimensions:

2.4.1.1 Level of Visual Refinement

The prototypes created in ProtoActive are sketch based with a low level of refinement. As mentioned by Buxton [24], prototypes with a low level of detail and refinement encourage users to provide more feedback. In later stages of design, ProtoActive can be used in collaboration with image editors and tools to create high-fidelity prototypes to explore the functionality and present flow between pages of the prototype. This functionality is better explained in section 4.3.1.7.

2.4.1.2 Breadth of Functionality

Different breadth of functionalities can be covered with ProtoActive as it allows designers to easily create several pages in the prototype that can cover a wide range of functionality.

2.4.1.3 Depth of Functionality

Different depths of functionality can also be achieved with ProtoActive. In the same fashion that a whole application can be prototyped using ProtoActive, a single task or behavior can be designed and evaluated using ProtoActive.

2.4.1.4 Richness of Interactivity

In the design of touch-based applications, ProtoActive is heavily based in interactivity as it allows designers to create their own interaction techniques through custom gestures and use them to interact with the prototype, simulating the behavior of the application through page transitions triggered by gestures.

2.4.1.5 Richness of Data Model

No mechanism to communicate with any form of data source was provided to keep designers from spending too much time with the prototypes in details such as populating data sources.

2.4.2 The Anatomy of Prototypes

The study by Lim *et al.* [21] focuses on the support for design exploration in the prototype. Their studies reveal two key dimensions: prototypes as filters and prototypes as manifestations. Their framework is based on two fundamental aspects of prototypes: prototypes leading the creation of meaningful knowledge about the final design as envisioned in the design process; or prototypes as manifestation of design ideas. Here this difference is explained in prototypes as filters or prototypes as manifestations.

2.4.2.1 Prototypes as Filters

The term filtering is used to imply that a designer uses prototypes to screen out certain details and put emphasis in others. ProtoActive allows designers to create interaction based prototypes, where the design and visual details can be set apart and the gestures used to interact with the application can be evaluated. The author of this thesis recommends that the study by Hinrichs and Carpendale [5] is considered, which suggested that interaction design should take into consideration the context in where the gestures used to interact with the application occur.

2.4.2.2 Prototypes as Manifestations

The manifestation of a prototype can be determined considering the economic principle of prototyping: *"the best prototype is one that, in the simplest and most efficient way, makes the possibilities and limitations of a design idea visible and measurable"* [37]. On the light of this definition, by allowing designers to evaluate their prototypes in the device where the final application will be used, ProtoActive makes clear the constraints and possibilities of a design idea, taking advantage of aspects of the device such as size constraints, orientation of the device and interaction possibilities. This allows users to have a good grasp of the experience the final product will offer.

CHAPTER THREE : RELATED WORK

This thesis proposes a process for designing ITS applications and presents a prototyping tool coupled with a gesture learning tool that together allows designers to define gestures and evaluate them in early stages of the design process with no programming effort. This chapter presents the current research in the following fields:

- design of multi-touch applications;
- research in prototyping;
- existing tool support for prototyping;
- gesture recognition and definition with emphasis on tools to aid the definition of custom gestures;
- design of tangible applications, its problems and tools to help designers.

The following sections in this chapter will show the related work in the fields mentioned above.

3.1 Design of multi-touch applications

Hesselmann and Boll propose *Surface Computing for Interactive Visual Applications* (SCIVA), a user-centered and iterative design approach addressing some challenges in designing ITS applications [3]. Their design process gives a general overview of the most important aspects in design of ITS applications. The solution in this thesis provides a tool suite that allows designers to follow three steps of the SCIVA design process: defining manipulation functions, conducting user studies to create gestures and evaluating the system with the user to detect flaws from previous steps. Also studying ways to interact with tabletops, Hinrichs and Carpendale found in 2011, that the choice and use of multi-touch gestures are influenced by the action and social context in which these gestures are performed, meaning that previous gestures and context influence the formation of subsequent gestures [5]. Also supporting the contextualization of interaction is Krippendorff [23] highlighting that design is not only about making things but also about making sense of things. Both studies suggest that to evaluate gestures it is necessary to contextualize them in the scenario that they will be used.

North *et al.* [34] studies how users interact with objects in multi-touch surfaces and how designers can create intuitive and natural gestures. They start from the assumption that interacting with objects on a multi-touch surface is an experience closer to manipulating physical objects on a table than using a desktop computer with keyboard and mouse. They study whether familiarity with other environments influences how users approach interaction with a multi-touch surface computer as well as how efficiently users complete a simple task. Among their findings, they show that users who started with the physical model had a better performance when accomplishing the task on the surface, which supports their initial assumption, but they also suggest that more complex gestures, (e.g.: using two hands for a selection) might not work well in a surface tabletop, meaning that there should be a balance between physical metaphors and supporting gestures to invoke automation.

Trying to understand users' preferences for surface gestures, Morris *et al* [35] compare two gesture sets for interactive surfaces: one created by end-user elicitation and one authored by three HCI researchers. The study used the feedback of 21 participants on 81 gestures which were previously created by a mixture of end-users and HCI researchers. Their results showed three main findings:

- their participants had similar gesture preference patterns;
- these preferences were towards physically and conceptually simple gestures;

- these simple gestures had been designed by larger sets of people, even though participants did not know how many authors created the gesture.

Their findings suggest that participatory design methodologies should be applied to gesture design, such as the user-centered gesture elicitation methodology.

Studying the inconveniences that can be generated by touch based interactions, Gerken *et al.* [36] focuses on how users compensate for conflicts between non-interactivity and interactivity created by unintended touch interaction when using a multi-touch enabled tabletop. They conclude that touch-enabled devices can lead to "*touch-phobia*", reducing pointing and leading to less efficient and fluent communication. They suggested solution is to make touch smarter and more context-aware.

Norman and Nielsen [2] published a usability study that highlights the new concerns that should be addressed by designers when creating new touch-based interfaces and ways of interacting with them. The authors propose a balance between creative means of interacting while preserving basic HCI principles, but guidelines for processes that can help designers follow a user centered design approach in the development of ITS applications are limited [3]. Hence, there needs to be an objective way to evaluate the usability of gesture-based applications that can be conducted in early stages of the design preserving HCI principles and having users involved in early stages of the design, helping designers follow a user centered approach.

3.2 Research in prototyping

Holzman and Vogler [37] studied the problem of prototyping applications that have a major focus in interactivity. They state that due to mobile applications being used in a rich real world context that for example affects how a user holds the phone or how many fingers a user can use, paper prototypes do not provide a good emulation of the scenario. However high-fidelity prototypes developed for the mobile device are costly and require too much effort for a prototype. They propose a system that allows the creation of sketch prototypes for mobile applications. Despite their focus in interactivity and taking advantage of the device, their prototyping approach does not allow designers to try different gestures that do not come built in the prototype. A similar study with designing mobile applications [38] also suggests that the prototypes should follow the size constraints of the device the application it will run on. Despite ProtoActive currently not being supported by any mobile device, evaluating the prototypes with the constraints of the target device was also considered when developing ProtoActive. This allows designers to have a real-size control design allowing users to evaluate the prototypes within the size constraints of the final application.

Unger and Chandler [39] explain the benefits of paper prototyping as a low-cost, fun and effective way to get user feedback and uncover design-related issues before being heavily invested in the project. For digital prototypes, the authors describe the outcome of these digital prototypes based on the timeline, the audience (or who will use the prototype) and what type of resources, tools and skills are available to create the prototype. The timeline will define how much effort and time a team wants to put into prototyping. By knowing your audience, a team can define specific parts of the software to prototype and where the prototype should the focus (e.g.: depending on task to be tested, does it matter if the prototype is black-and-white or if it actually reflects the colors of the intended final application?). Finally, the type of resources and skills will set the constraints for which tools a designer will use to create a prototype. Considering these factors, a digital prototyping tool will benefit a wider range of designers by:

- allowing the creation of rapid sketches that can fit into time-constrained projects;

- giving designers the chance to focus in different aspects of a prototype, for example the visuals or the interactions;
- focusing in its learnability by not requiring expert skills such as software programming in order to create prototypes.

The potential differences between paper-based or computer based low-fidelity prototypes are also studied by Sefelin *et al.* [18]. Their study leads to two main results: paper and computer-based prototypes lead to almost the same quality and quality of user statements; and subjects in general prefer to evaluate computer prototypes. The second result suggests that computer-based low-fidelity prototypes should be the one to always use, keeping in mind how comfort the participant is, as this is one of the major factors of a successful usability study, but the authors claim three points on where papers should be a preferable medium for prototyping:

- when there are no available prototyping tool to support the ideas;
- when the tool requires expertise that members of the design team might not have;
- when tests should lead to a lot of drawings.

Based on these results, the advantages of the paper prototype over a computer-based low-fidelity prototype can serve as requirements for a prototyping tool that aims to have the benefits of low-fidelity prototyping. This tool should allow the expression of the ideas and customization from the designers and require minimum expertise to use. A low-fidelity prototyping tool should also allow participants to easily sketch over the interface as a medium of feedback during evaluation.

The importance of creating prototypes for ITS applications is shown in a study by Derboven *et al*. in 2010 [22]. Their study introduces two low-fidelity prototype methods for multi-touch surfaces. In comparison to this thesis, their approach consists of "physical" materials such as paper, cardboard and markers, while our approach proposes an ITS tool, allowing users to evaluate the prototypes on the device the applications are designed for as well as the custom interactions that will be used in the final application.

The following sections will discuss prototyping tools created in academia and published as scientific papers.

3.2.1 CrossWeaver

CrossWeaver [40] allows a user interface designer to sketch storyboard scenes on the computer, specifying command transitions between scenes. *CrossWeaver* also allows scenes to target different output devices, this means that prototypes can run across multiple standalone devices simultaneously, processing multimodal input from each one this allows for a wider variety of inputs. The prototypes can be interacted via keyboard, mouse, stylus pen or speech commands. It also allows designers to track the interactions of users evaluating the prototype. One of the drawbacks is the lack of customization as the interactions that are possible to evaluate are the pre-built interactions that come with *CrossWeaver* (with the exception of using wizard of Oz recognition).

3.2.2 Raptor

Raptor is a sketching tool for tabletop games [41]. *Raptor* shows how tabletop interaction can support game design by enabling collaboration in the design and testing process. One of the drawbacks of the tool is that it requires a preparation process in order for a designer to be able to get the benefits from the tool. It requires a software developer to create a library of the objects used in the gameplay to allow designers to create prototype of games with them (e.g.: in their study, a 3D model of a racing car), which adds cost and effort to the prototyping stage. Also, raptor was built to enable designers to prototype a specific subset of tabletop games, this limitation precludes games with large amounts of vertical motion or occlusion from above.

3.2.3 Sketchify

Obrenovic and Martens propose *Sketchify*, a tool for sketching user interfaces [42]. *Sketchify* combines elements of traditional freehand sketching functional extensions and end-user programming tools, such as spreadsheets and scripting. *Sketchify*'s approach choose to cover a wide range of different technologies by allowing designers to interact with these technologies during prototype. This approach allows designers to control a wider range of technologies still in the prototype phase but comes at the cost of requiring the expertise to work with these technologies and textual programming from the designers.

3.2.4 UISKEI

UISKEI is a pen-based sketch prototyping tool that converts sketches into known *User Interface* (UI) widgets (e.g.: buttons and drop down lists). The designer can use the generated UI components to navigate through the pages in the prototype [43]. While *UISKEI* is a pen-based prototyping tool, it creates prototypes of WIMP applications while ProtoActive creates prototypes of gesture and tag based interfaces. A drawback from *UISKEI* is that it turns sketch interfaces into UI widgets which takes off what according to Buxton [24] is one of the benefits of sketch-based prototypes is that the unfinished look of the interface allows users to focus in what the designer wants to evaluate and encourages users to provide more feedback.

3.2.5 SILK

SILK (Sketching Interfaces Like Krazy) [44] was an early prototyping tool developed by James Landay in 1996. *SILK* is a sketch-based user interface design tool where a designer can sketch various states of the user interface, to specify the behavior of the interface. The sketches are assembled together into a storyboard. It also incorporates widget recognition, which means that when a designer draws a sketch similar to a button, *SILK* will recognize it and turn it into a button. By using the storyboard to visualize all the pages, by drawing an arrow from a user interface object in one sketch to another sketch page, the designer says that, in the user interface, if the user clicks on the object, the program will transition to the screen shown at the end of the arrow. The functionality to turn sketches into UI widgets has the same drawback as explained in *UISKEI* (section 3.2.4). *SILK* also does not allow the evaluation of the input of the user, only evaluating the output of an interaction (e.g.: turning pages when user clicks).

3.2.6 DENIM

DENIM is a sketching tool for prototyping web and desktop applications [45]. It allows designers to free-hand draw on canvas and also provides custom drag and drop controls that can be used. Its main feature is to allow designers to change visualization of perspective by having a *zoomable* canvas which gives designers different perspectives of the prototypes, from a really detailed level to an overview of the pages, giving a good sense of the flow between the pages. Similar to *SILK* (section 3.2.5) it only evaluates the output of the user interaction and not the interaction itself.

3.2.7 DEMAIS

DEMAIS (Designing Multimedia Applications with Interactive Storyboards) [46] is a sketch-based, interactive multimedia storyboard tool that uses ink strokes and textual annotations as an input design vocabulary. By operationalizing this vocabulary, the tool transforms an otherwise static sketch into a working example. The behavioral sketch can be quickly edited using gestures and an expressive visual language. In a similar fashion as *SILK* (see section 3.2.5) it also analyzes ink strokes to turn into widgets, but it also tries to recognize the stroke as a behavior. It also contains a narration feature that allows designers to leave comments in sketches that can be listened to during evaluation. *DEMAIS* lacks the support to explore interaction as gesture or tangibles as it focuses in visualizing multimedia, adding a layer of complexity for the usability of the tool.

3.2.8 Active Story Touch

ProtoActive is based on *Active Story Touch* (AST) a tool developed by Hosseini-Khayat *et al.* that targets the creation of low-fidelity prototypes for touch-based applications [47]. AST had to be modified in order to cover the needs of an ITS application and to incorporate custom gestures.

3.3 Existing tool support for prototyping

Besides the efforts in academia to improve prototyping, there are several prototyping or sketching tools available for designers to use. In order to create a list, we crossed the results from four HCI websites [48, 49, 50 and 51] that had ranks of prototyping tools and included in this list the five most cited ones and compared them to our solution. In addition, trying to avoid bias from price affecting the choice of the tools by the websites, the top ranked results in a Google search of the term "*prototyping tool*" were added to this list.

3.3.1 Balsamiq Mockups

Balsamiq Mockups [52] is a web-based tool that rapidly creates wireframes, and exports the created wireframes to PDF or PNG files. It offers designers pre-made controls with a *sketchy* interface that helps to rapidly create the prototype of an interface. To help finding controls, *Balsamiq Mockups* divides controls into categories, including an iPhone category with interface elements that can be commonly found in iPhone applications. It also offers skin customization for sketches and wireframes. But it does not allow designers to free-hand draw controls limiting creativity and it lacks creating multiple pages to simulate system behavior.

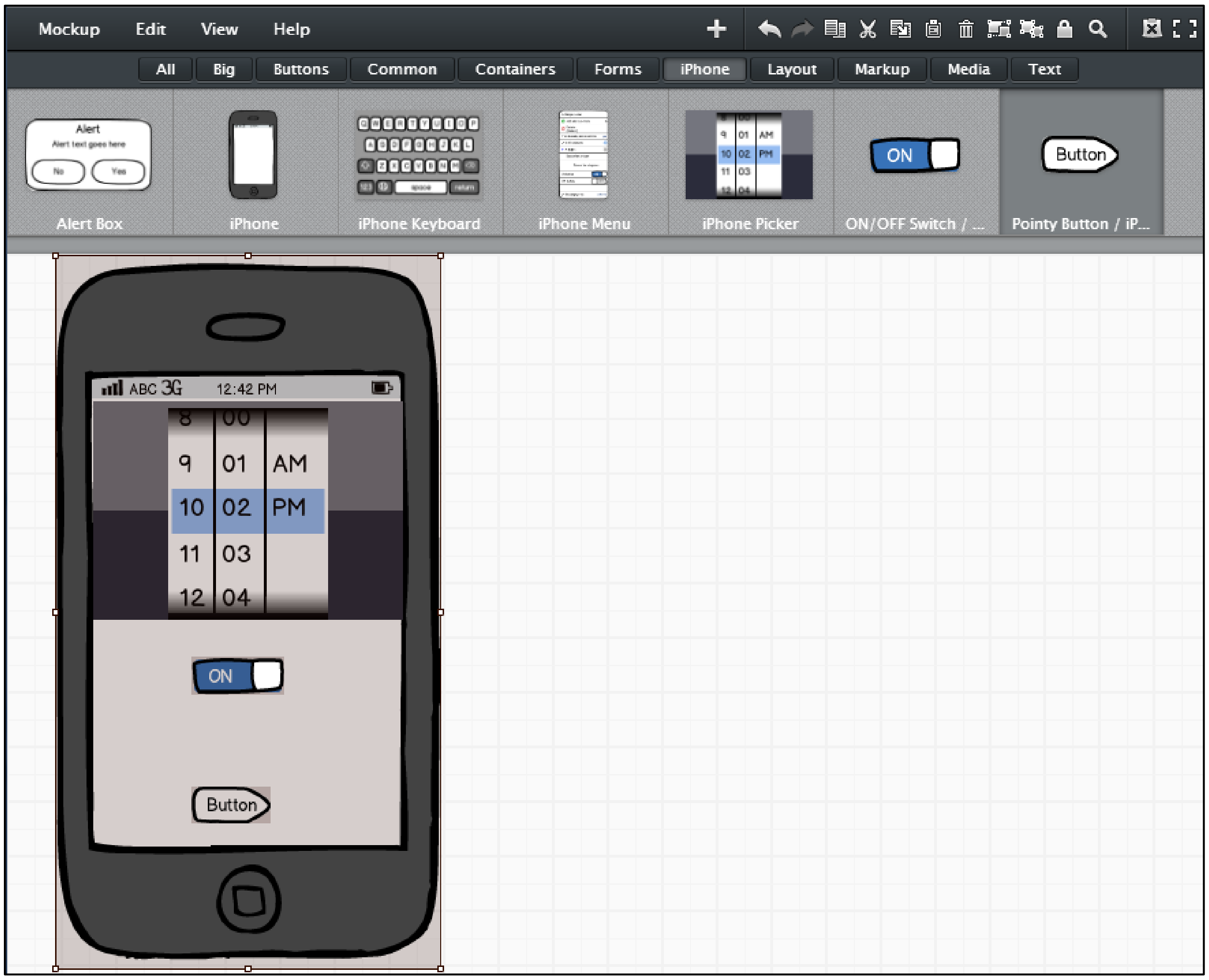


Figure 4 Using iPhone controls in Balsamiq Mockups [52]

3.3.2 *Pencil*

Pencil [53] is a Mozilla Firefox add-on to that helps designers with GUI prototyping and simple sketching of web applications. It allows inter-page linking and allows designers to export their prototypes to several formats, including HTML, PNG and PDF. It uses a drag and drop widget feature with the possibility to have sketchy controls in a fashion similar to the one used in *Balsamiq Mockups* (see section 3.3.1). It lacks support for extensive evaluation of interaction as it only allows the page movement to be triggered by mouse clicking. It has an easy setup as it can be used as a Firefox add-on or standalone application, but it does not allow free-hand sketching or interaction through the specific capabilities of a device (e.g.: fiduciary markers detection on the Microsoft Surface or multi-touch gestures).

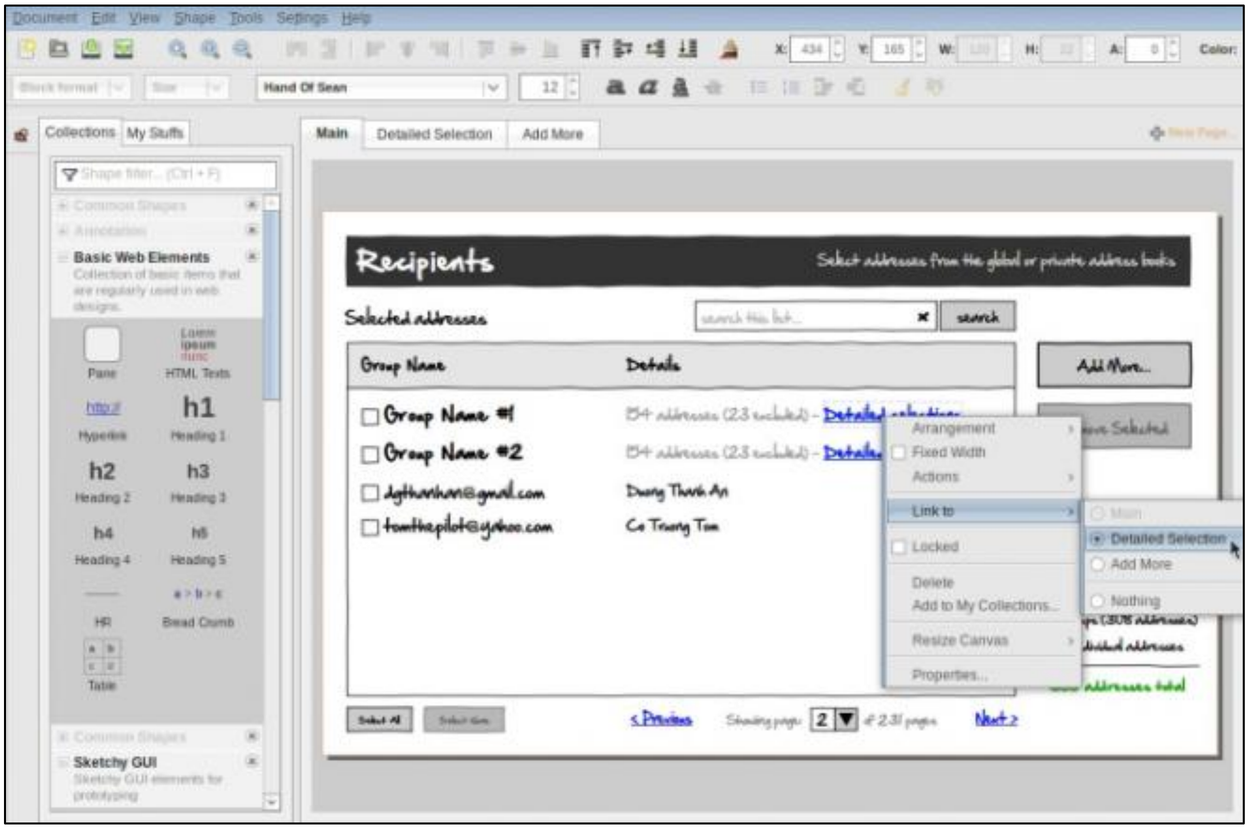
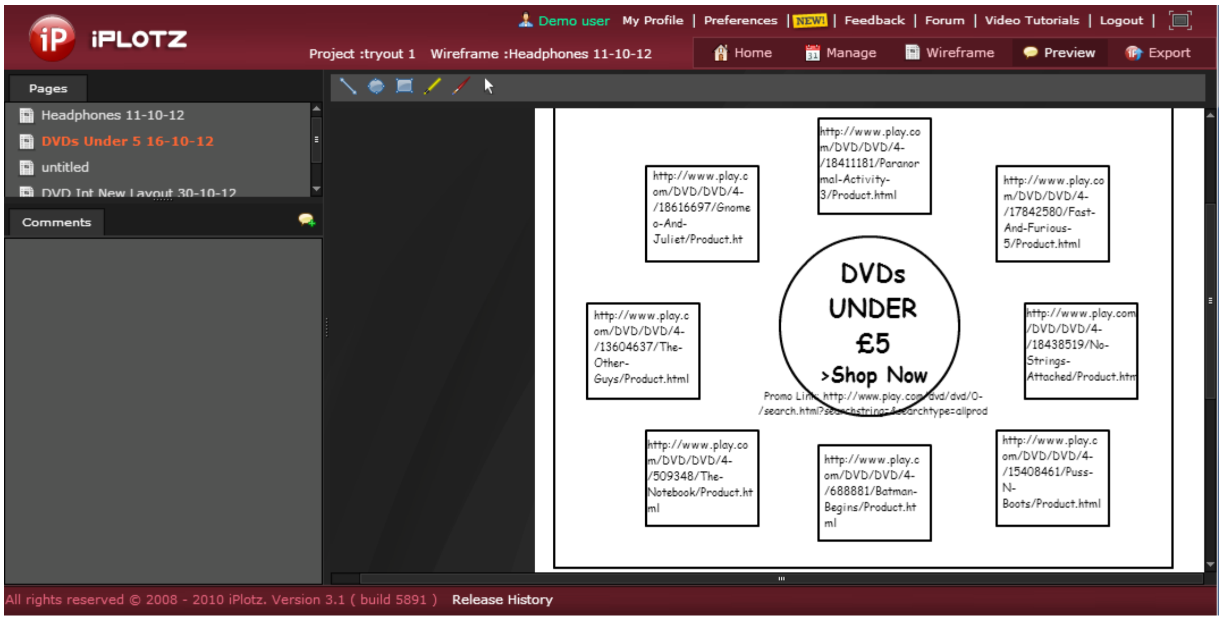


Figure 5 Pencil GUI interface with page linking, extracted from [53]

3.3.3 iPlotz

In *iPlotz*, designers can create clickable, navigable wireframes of websites or software applications [54]. *iPlotz* also has a manager feature for designers and developers to build the project. A designer can easily change the appearance of the prototype by switching between sketchy, Windows or Mac appearance. It allows the creation of interactive prototypes, but only through mouse click events that trigger page transitions, not allowing the evaluation of touch-based interactions. *iPlotz* also lacks the support for free-hand drawing of interface, making the designer limited by the widget options or to use lines or a set of shapes.


iPLOTZ
Demo user | My Profile | Preferences | NEW! | Feedback | Forum | Video Tutorials | Logout
Project :tryout 1 Wireframe :Headphones 11-10-12
Home | Manage | Wireframe | Preview | Export
Pages
Headphones 11-10-12
DVDs Under 5 16-10-12
untitled
DVD Int New Layout 30-10-12
Comments
DVDs UNDER £5 >Shop Now
http://www.play.com/DVD/DVD/4-/18411181/Paranormal-Activity-3/Product.html
http://www.play.com/DVD/DVD/4-/18616697/Gnome-o-And-Juliet/Product.html
http://www.play.com/DVD/DVD/4-/17842580/Fast-And-Furious-5/Product.html
http://www.play.com/DVD/DVD/4-/13604637/The-Other-Guys/Product.html
http://www.play.com/DVD/DVD/4-/18438519/No-Strings-Attached/Product.html
http://www.play.com/DVD/DVD/4-/509348/The-Notebook/Product.html
http://www.play.com/DVD/DVD/4-/688881/Batman-Begins/Product.html
http://www.play.com/DVD/DVD/4-/15408461/Puss-N-Boots/Product.html
All rights reserved © 2008 - 2010 iPlotz. Version 3.1 (build 5891) Release History
Figure 6 iPlotz interface using sketchy appearance, extracted from [54]

3.3.4 *AxureRP*

AxureRP [55] can generate interactive HTML pages for user evaluation. However, it provides considerably more complex interactions than those provided in other HTML-based prototyping tools. It works with UI widgets, not allowing designers to free-hand sketch interfaces, which can be seen as a drawback as it limits designers to create prototypes that contain only pre-built widgets. It allows designers to create low, medium and high-fidelity prototypes. However, in order to create low-fidelity prototypes, the designers can’t use free-hand sketches and are constrained to use UI widgets with a sketchy look (as seen in Figure 7), which limits also the creativity as the design can only use controls that are pre-built with the tool. For interaction, the widgets on canvas have events that can be triggered depending of the widget (e.g.: an image widget has a mouse click, mouse enter, mouse out event, while a button only has a mouse click event). A pre-built set of actions can be triggered, adding another layer of complexity that escapes from the simple page transition used in papers. It is destined to create prototypes of websites, having a different focus than our solution, but it lacks the support for touch or gesture-based interactions that can already be used in browsers, depending of the device support.

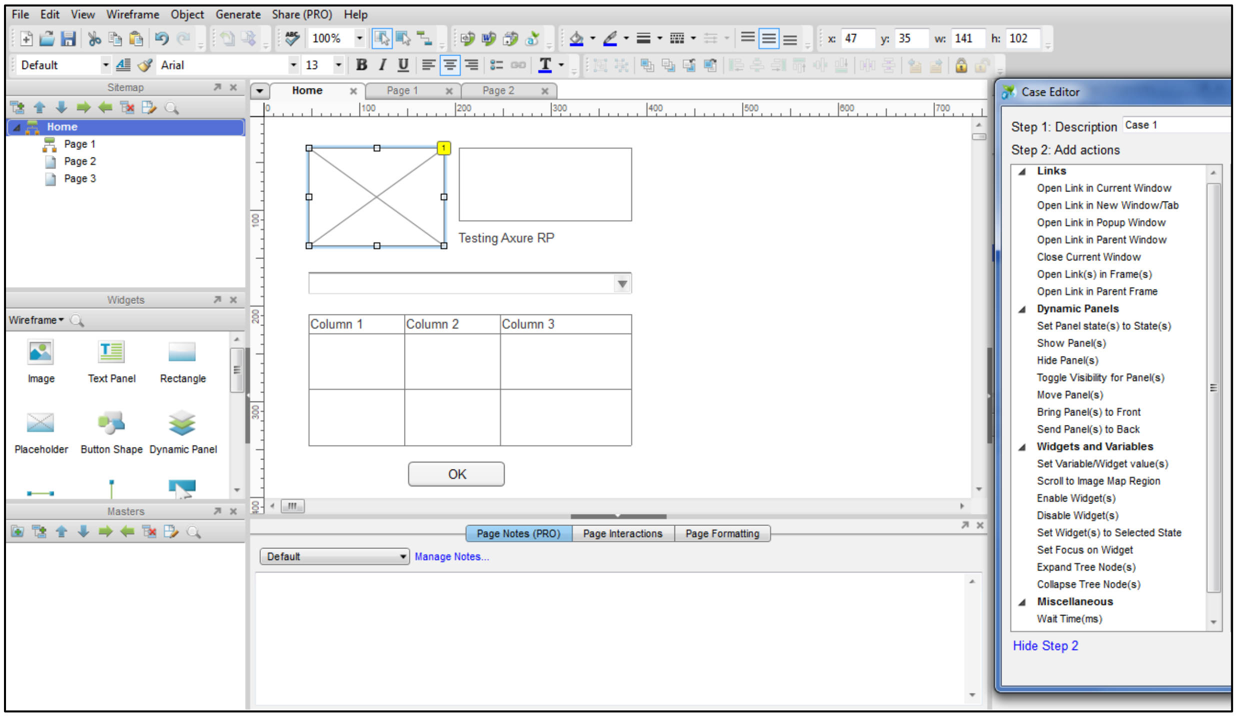


Figure 7 Axure RP showing a prototype page and the actions tab

3.3.5 Mockingbird

Mockingbird [56] is an online tool that makes it easy for designers to create mock-ups of applications, link them together to preview and interact with them. It is a drag and drop based prototyping tool that has widgets with a sketchy appearance. As an advantage over the other drag and drop prototyping tools, it comes with a non-standard set of widgets, like maps or video widgets, but does not allow designers to free-hand draw their interface, constraining creativity. It comes with a functionality to duplicate pages, helping to simulate behaviors with page transitions, and it does not allow designers to evaluate the interactions with the prototypes.

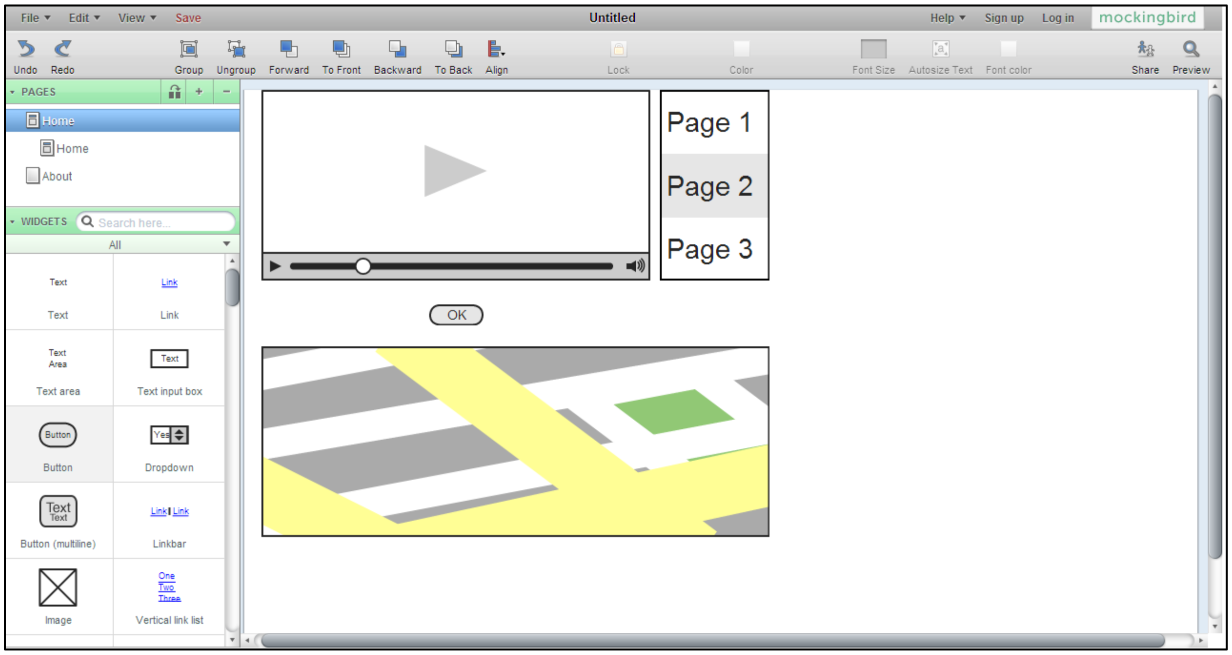


Figure 8 Mocking bird screen with a map and a video player widget

3.3.6 *Microsoft Sketch Flow*

Microsoft Sketch Flow [57] provides support for sketching user interface designs, as well as interactive *WPF* controls. It allows designers to create wireframes using standard *WPF* controls. *Sketch Flow* has an option to add controls with a *sketchy* look, trying to give a low-fidelity prototype idea. It also has a feedback area where a user can provide some comments and a map area where the whole flow of the pages in the prototype can be seen. It does not allow designers to create free-hand sketches thus limiting the creativity. As it allows the creation of low-fidelity and high-fidelity prototypes in the same tool with no distinction, it might mislead designers in spending too much time on details and polishing its visual for low-fidelity prototypes which is one of the advantages of low-fidelity prototypes (see Table 1).

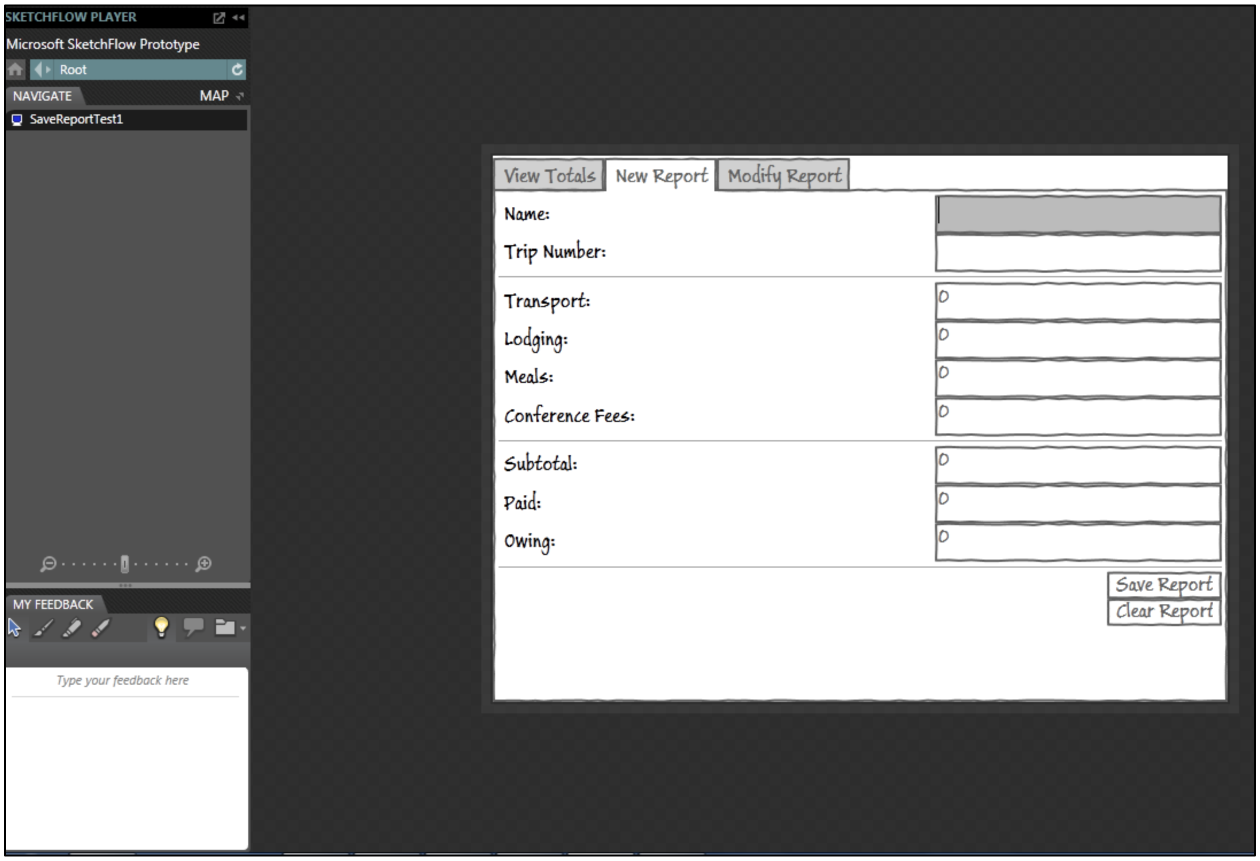


Figure 9 Microsoft Sketch Flow

3.3.7 Fore UI

ForeUI [58] is a UI prototyping tool, that similar to *iPlotz* (see section 3.3.3) allows designers to easily change the appearance of their prototypes, varying for a sketchy looks to more refined prototypes. It allows designers to use drag and drop widgets and to customize and store widgets that can be used among different projects (e.g.: a custom button with different background color and size). In a similar fashion to *iPlotz* (see section 3.3.3), it allows designers to create customization and different shapes by allowing designers to drag lines onto the canvas. This solution is counter-productive to simply free-hand draw. Designers can design the behavior of prototypes by defining events on specific widgets to be used in simulation. It only allows a predefined list of events that are mouse or keyboard based, which makes this tool only valuable to evaluate the interaction of prototypes of *WIMP* applications. As an advantage, it allows the prototype to be exported to wireframe images, PDF documents or HTML5 simulation, making it easier to have users to evaluate it.

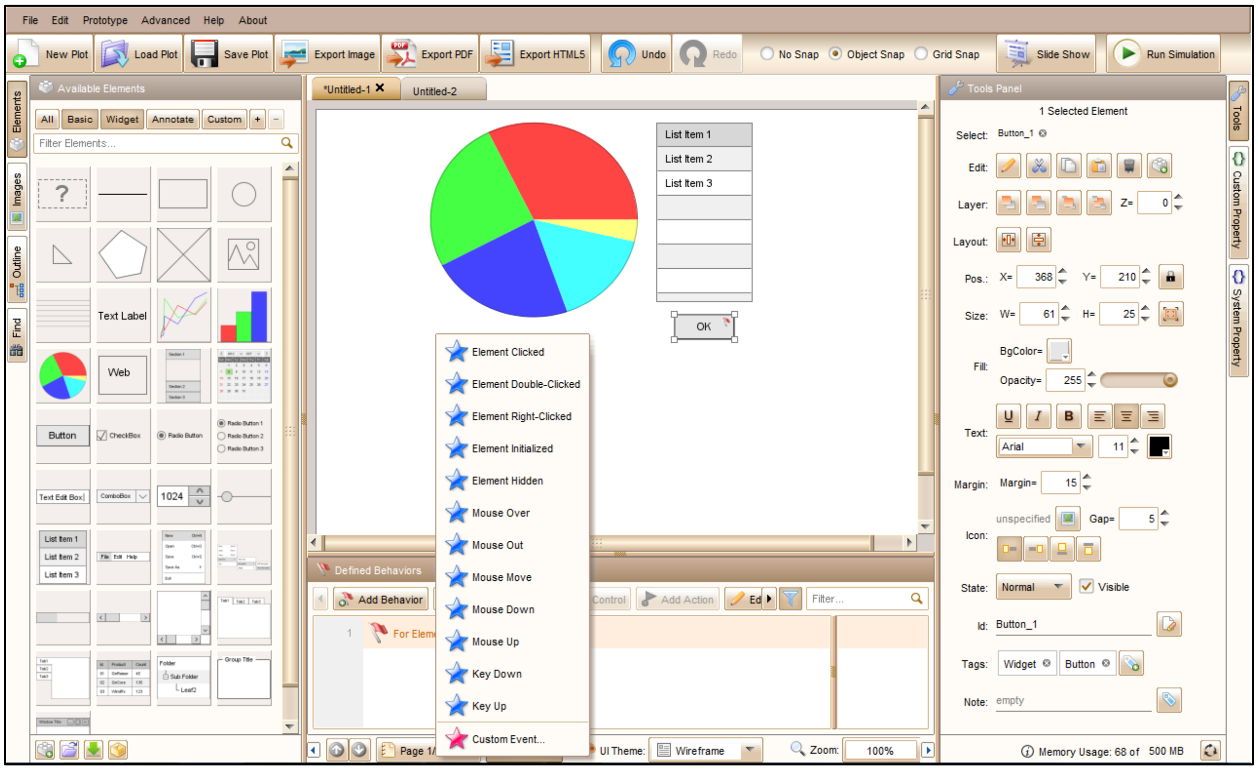
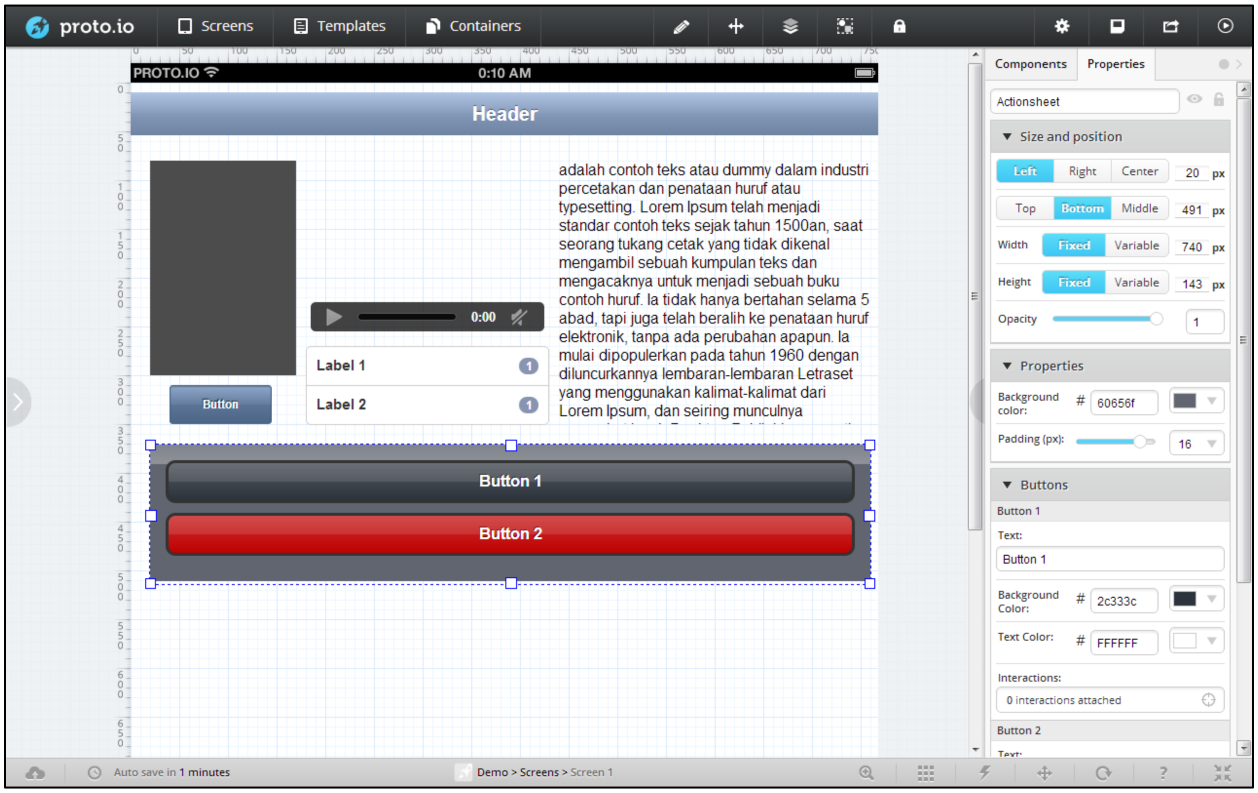


Figure 10 Fore UI showing the list of possible events for a button widget

3.3.8 *Proto.io*

Proto.io [59] is a UI prototyping tool tailored for mobile and tablet applications. It was designed to be used in a web-based environment allowing designers to start by creating a project for mobile phones or tablets, having the device constraints already built in. It allows designers to link pages together by using a predefined set of actions that are custom to hand held devices allowing designers to simulate interactivity such as clicks, taps, tap & holds as well as swipes. It also allows designers to simulate transitions such as slides, pops, fades and flips, creating a closer resemblance to the real experience through the prototype. It comes with a predefined set of gestures that does not allow to be extended by custom gestures created, which might restrict the evaluation of the interactions in the prototype. Proto.io also works based on drag and drop widgets, constraining creativity and allowing designers to only create prototypes with known widgets. In a similar fashion to our solution, it allows designers to define interaction areas that can be triggered by a list of events without being attached to any widget.



adalah contoh teks atau dummy dalam industri percetakan dan penataan huruf atau typesetting. Lorem Ipsum telah menjadi standar contoh teks sejak tahun 1500an, saat seorang tukang cetak yang tidak dikenal mengambil sebuah kumpulan teks dan mengacaknya untuk menjadi sebuah buku contoh huruf. Ia tidak hanya bertahan selama 5 abad, tapi juga telah beralih ke penataan huruf elektronik, tanpa ada perubahan apapun. Ia mulai dipopulerkan pada tahun 1960 dengan diluncurkannya lembaran-lembaran Letraset yang menggunakan kalimat-kalimat dari Lorem Ipsum, dan seiring munculnya

Actionsheet

[✓ Left] Right Center 20 px
Top [✓ Bottom] Middle 491 px
Width [✓ Fixed] Variable 740 px
Height [✓ Fixed] Variable 143 px
Opacity 1

Background color: # 60656f
Padding (px): 16

Button 1
Text: Button 1
Background Color: # 2c333c
Text Color: # FFFFFF
Interactions: 0 interactions attached

Figure 11 Proto.io interface using iPad widgets

3.4 Comparing features of existing tools

Based on the prototyping tools mentioned in sections 3.2 and 3.3, Table **2** compares these tools features: the type of interaction, the level of customization, the features for drawing and any special features.

Table 2 Prototyping tools comparison

Tool name	**Interactions**	**Customization**	**Drawing feature**	**Additional**
CrossWeaver	Keyboard, mouse, stylus pen or speech commands	pre-built only	Free-hand sketch and shapes	Allows multi-device interaction
Raptor	Joystick based	Pre-built only	3D design using gesture to manipulate scenarios	Requires programming phase, can be used in tabletops
Sketchify	From different I/O devices	Programmable	Free-hand sketch	Allows communication with different devices but requires programming expertise
UISKEI	Mouse click only	Pre-built only	Converts sketches into pre-built widgets	
SILK	Mouse click only	Pre-built only	Converts sketches into pre-built widgets	

DENIM	Mouse click only	Pre-built only	Free-hand and draggable widgets	
DEMAIS	Mouse click only	Pre-built only	Converts sketches into pre-built widgets	Allows designers to leave comments in sketches that can be listened to during evaluation
Active Story Touch	Gestures	Pre-built only	Sketches and draggable widgets	Served as a basis for ProtoActive
Balsamiq Mockups	Mouse click only	Pre-built only	Draggable widgets	
Pencil	Mouse click only	Pre-built only	Draggable widgets	Browser add-on
iPlotz	Mouse click only	Pre-built only	Shapes and draggable widgets	Different fidelity levels
AxureRP	Mouse and keyboard based events (e.g.: click, drag…)	Pre-built only	Draggable widgets	Different fidelity levels

Mockingbird	Mouse click only	Pre-built only	Draggable widgets	Different fidelity levels and with non-standard set of widgets
Microsoft Sketch Flow	Mouse and keyboard based events (e.g.: click, drag…)	Pre-built and programmable interactions	Shapes and draggable widgets	Prototypes can be turned into working applications
Fore UI	Mouse and keyboard based events (e.g.: click, drag…)	Pre-built only	Shapes and draggable widgets	Different fidelity levels
Proto.io	Mouse click and gestures	Pre-built only	Draggable widgets	For mobile and tablet
ProtoActive	Gestures	Pre-built and custom	Free-hand sketch	Can be used in tablet and tabletops; allows the addition of images

3.5 Gesture recognition and definition

Wobbrock *et al.* published in 2009 a study to gather the gestures that people would use for specific tasks [6]. For the same tasks, they compared gestures created by HCI experts and users.

The result showed the gestures created by the experts did not cover half of the ones created by users, meaning that designer opinion or expertise alone is not sufficient and user opinions are also required. This enforces the need for a user-centered design process when defining the gestures that will be used to interact with the application. In the same study, users are presented with an animation showing an application behavior and are then asked which gesture the users think should be used to trigger the respective behavior. This study also proposes a taxonomy of surface gestures and a user-centered gesture set generated based in their study that included 1080 gestures. One of the drawbacks of the study, as stated by the authors themselves, is that the application context could impact on the choice of gestures by the user, as could the larger contexts of organization and culture. This drawback is one of the motivations to ProtoActive, as it allows designer to evaluate the gestures in the context of the application they will be used in, by allowing the evaluation of the gestures in prototypes of the final application.

A statistical modeling approach, namely *Hidden Markov Models* (HMM), is used by Damaraju and Kerne [60]. They propose a system that uses video processing and HMM to learn gestures. Similarly to the approach proposed in this research, their system learns gestures by examples, however, while their system detects only finger gestures, our approach also detects and combines hands and tags in a single gesture definition.

GISpL [61] is a gesture specification language that allows the description of the behavior of device-independent gestures using *JSON* syntax [62]. Languages to describe gestures have been widely researched with examples including the study by Scholliers *et al.* [63], Kammer *et al.* [64], Khandkar and Maurer [10] or $N proposed by Anthony and Wobbrock [65]. These languages provide the necessary tools and abstractions to help developers create custom gestures, but creating a new gesture definition usually requires some understanding of programming which leaves designers out of the creation process of custom gestures and only allows users to give feedback about the interactions late in the design process.

Some studies focus on gesture recognition and gesture definition as a specific problem. In 2007, Mitra *et al.* [29] published a survey that covers several approaches used in gesture recognition and in 1997 Bobick *et al.* [66] proposed a state-based approach to the representation and recognition of gestures.

In 2010, Khandkar and Maurer proposed a *Gesture Definition Language* (GDL) to be used by developers to define new gestures and also created a *Gesture Toolkit* [10]. This thesis presents an extension of this work as it adds three main features to the domain-specific language defined by Khandkar and Maurer:

- recognition of the image of the BLOBs in order to allow the recognition and definition of hand gestures;
- adds a new type of primitive value, which allows primitives to have their value based on the value of other primitives.

The contributions of the tool regarding the *Gesture Toolkit* are explained in more depth in Chapter Five. Also in 2010, Khandkar *et al.* proposed a tool to support testing complex multi-touch gestures [67]. This enables automated testing for gesture-based applications. The testing tool by Khandkar focuses on how accurate a system recognizes a gesture while my work answers the question of how useable the gesture actually is for the application it will be used on. This solution differs from the one proposed in this thesis as it focuses more on unit testing (i.e. software engineering) while the solution proposed in this paper focuses on sketch-based prototype evaluations which involve users. In a similar fashion as the approach in this thesis, there are tools that provide users with user-friendly ways for creating gestures.

A gesture recognition toolkit, *GART,* is described by Lyons *et al.* [13]. The toolkit provides an abstraction level to the machine learning details, helping software developers to create gestures. This approach allows the use of different machine learning algorithms as well as different sensors (mouse, camera, Bluetooth accelerometers). One of *GART* drawbacks is that it does not allow continuous gesture recognition or to work with multi-touch devices.

PROTON is a tool by Kin *et al.* that uses regular expressions to define gestures [14]. The authors present an editor to help create the regular expressions as well as an input *ambiguity* and *callback* handling mechanism. *PROTON* does not allow the definition of gestures that contain directions, being impossible to define "*Left Swipe" and "Right Swipe"* as different gestures. Both *GART* and *PROTON,* require software development expertise to create gestures which adds time and cost, leaving users to only evaluate the gestures late in the process of designing the application.

The machine learning approach used in this paper is also explained in a study published by the author of this thesis [68] and is similar to that applied in 2007 by Cottrell *et al.*, who presents a proof-of-concept plug-in to Eclipse for automatically determining correspondences as an early step in a generalization task [69].

Instead of having a user training the tool, the paper in 2009 by Freeman *et al.* [70] proposes a tool to help users learn multi-touch and whole hand gestures on a tabletop surface. The users learn by studying an example provided by the tool and obtaining feedback on their attempted gestures.

3.6 Designing tangible applications

Studying the benefits of prototyping TUI applications, Wiethoff *et al.* proposed a method called *Sketch-A-TUI*, which allows users to create lo-fi 3D paper objects that can be recognized by a capacitive surface using conductive ink in the paper objects to simulate touch points [16]. Different than *Sketch-A-TUI*, ProtoActive does not require any programming effort, and tangibles can be used in prototypes of the scenario they will be used, as suggested by Hinrichs and Carpendale in a study using interactive tabletops [5].

In a similar fashion as *Sketch-A-TUI,* Yu *et al.* presented *TUIC*, a technology that consists of three approaches to simulate and recognize multi-touch patterns using both passive and active circuits embedded inside objects [71]. *TUIC* only provides the technology to generate tangibles, not providing designers with a way to evaluate the tangibles in the applications they are designed to be used in.

The study by Esteves considers the design of TUI applications problematic and a challenging ad-hoc process that requires the adoption of guidelines and methodologies if the area continues to grow [72]. The paper considers theories of embodied cognition as source for guidelines that are suitable for designing tangible interactions. It also presents a toolkit that records and presents the interactions with tangible applications. This paper is a continuation of the toolkit proposed by Esteves and Augusto [73]. The toolkit logs the manipulation of tangible objects in order to create empirical methods for the study of tangible applications. The paper argues that the logs acquired by the toolkit can be used:

- to compare tangible interaction with other interaction paradigms;
- to compare among different tangible interfaces performing the same tasks;

- via integration into a structured design process. [73].

This thesis argues that a toolkit for recording manipulations of tangible interactions is a step towards developing a mature methodology to quantify and evaluate a tangible application. Whereas the toolkit proposed by Esteves and Augusto gives important information to help designers create TUI applications, this thesis offers designers with a tool that helps not only the design of tangibles and the tangibles interactions but also does it so in a sketch-based prototype scenario that can be achieved with minimum effort and in the scenario of the application where the tangibles will be used.

In a similar approach to this thesis, *DART-TUI* is proposed by Gandy *et al.* [74] and aims to make a particular class of TUIs accessible to a broader range of designers and HCI researchers by exposing TUI specific tools in a mixed-reality (MR) rapid prototyping environment. While *DART-TUI* offers designers an opportunity to create rapid prototypes using their toolkit, the complexity of the prototypes created suggests that they will receive a feedback similar to a high-fidelity prototype, only involving users late in the design process.

Papier-Mâché is a toolkit proposed by Klemmer *et al.* that helps developers create tangible applications. It abstracts the input hardware layer, allowing developers whom are not input hardware experts to develop TUIs [75]. It facilitates the implementation stage as it provides application developers a monitoring window that displays the current input objects, image input and processing, and behaviors being created or invoked. For simulating input when hardware is not available during implementation and debugging, it has a *Wizard of Oz* generation and removal of input. *Papier-Mâché* helps developers in the implementation stage of development which might lead to only involving users late in the design process.

CHAPTER FOUR : PROTOACTIVE

"*The low-fidelity prototype is informal and fast. No one will mistake it for the finished product. Because it is easy to change, stakeholders are willing to iterate, experiment, and investigate alternative ideas*" [30 p.88].

In order to help designers of ITS applications, this thesis proposes ProtoActive, a prototyping tool that:

- elicits user feedback through sketch-based prototypes that take in consideration the size constraints of an ITS application
- allows the evaluation of how users interact with the application by having prototypes that can be used through a pre-built set of gestures that can be expanded through a tool that allows the creation of custom gestures without requiring any programming effort.

ProtoActive is a sketch based prototyping tool for multi-touch devices that integrates with a gesture learning tool (IGT) to evaluate custom gestures in prototypes. This chapter will explain the process of designing ProtoActive involving requirements gathered from related work and from a qualitative study with participants from industry. The following sections will also explain ProtoActive features and the workflow of a designer using ProtoActive to create interactive prototypes.

To gather requirements for a sketch-based prototyping tool, the author used:

- existing research about computer-based prototyping tools and problems found in existing tool support for prototyping;
- a qualitative study that consisted of semi-structured interviews with five *User Experience* (UX) designers from different companies.

The two following sections will explain the design guidelines of ProtoActive that were gathered from drawbacks of the studied prototyping tools (section 4.1) and from a qualitative user study (section 4.2) with five UX designers from industry. Finally, the features of ProtoActive (section 4.3) are presented, incorporating the findings from the design guidelines.

4.1 Design guidelines based on drawbacks from related work

The existing research was discussed in sections 2.3, 3.2 and 3.3. The studies and tools discussed in these sections served to elicit requirements of a prototyping tool. For each study or tool discussed, the author highlighted potential drawbacks. The author derived requirements from the existing work that allow ProtoActive to overcome drawbacks that were identified in the previous chapter.

4.1.1 Improve the paper experience

Sefelin *et al.* [18] compares paper prototyping with prototyping using software tools. They study suggests three scenarios where paper prototyping would be a preferable medium. These scenarios are addressed by ProtoActive as follows.

4.1.1.1 Allow expression of ideas and customization from the designers

ProtoActive allows designers to create free-hand sketches on a drawing canvas. As mentioned in section 4.2.1, free-hand sketching allows designers to better explore their creativity.

4.1.1.2 Require minimum expertise to use

In order to simulate paper experience, ProtoActive should have an intuitive and easy-to-learn interface that allows designers to create prototypes without requiring much time to learn the application. In order to do so, the design of ProtoActive kept the functionality to the minimum required, having a sketching tool with basic commands found in any sketching tool: free-hand sketching, sketch eraser, color picker, and strokes selection. Additionally two features were added: adding and removing background which allows designer to import images into their prototypes. Finally to have a flow between the pages, designers can specify areas in the prototype page that when interacted with, will trigger a page movement set by the designer.

4.1.1.3 During evaluation, allow participants to easily sketch over the interface as a medium of feedback.

This guideline suggests two features:

- a designer should be able to save multiple copies of a prototype, so during evaluations users can suggest modifications on the prototype itself;
- the tool should be simple enough that making modifications to a page is as simple as sketching on a paper.

ProtoActive allows designers to save multiple copies of a prototype. By having a prototype per evaluation, a designer can make changes or even allow the user to make changes to pages of the prototype himself, during evaluation. As ProtoActive was designed to have a simple and easy-to-learn interface, drawing over a page can be easily accomplished by simply exiting evaluation mode and accessing the page the user wants to modify or add suggestions.

4.1.2 Help designers follow design guidelines for ITS applications

SCIVA [3] is an iterative process for designing gesture-based interfaces for interactive surfaces. In order to help designers have a more systematic approach in the design of ITS applications, ProtoActive helps designers follow three steps of the SCIVA design process: defining the right visualization, conducting user studies to create gestures and evaluating the system with the user to detect flaws from previous steps. The following sections explain how ProtoActive addresses these steps.

4.1.2.1 Defining the right visualization

In ITS applications there is a tight coupling between input (gestures and touch) and output (visualized objects on the screen). ProtoActive deals with this problem by allowing designers to create prototypes without constraining creative ideas by allowing the creation of free-hand sketch prototypes that will allow for any type of object on the screen. If designers decide to have a more accurate or consistent object visualization among different prototypes, ProtoActive allows designers to import pictures inside their prototypes. According to SCIVA, ProtoActive helps in two of the recommendations for this step: brainstorming and gathering feedback from users; and optimizing visualizations according to characteristics of ITS. The visualization optimization can be achieved by letting designers create and evaluate the prototypes in the ITS devices themselves. This allows for a realistic evaluation of distance, position and orientation of objects in the screen.

4.1.2.2 Conducting user studies to create gestures

For ITS applications, there are sets of defined gestures offered [9, 8] that help designers define the input of ITS applications. However these sets can be insufficient due to particularities of devices, location, context and orientation. In order to improve the user experience, it is necessary to evaluate the interaction of ITS applications with users. ProtoActive gives designer a pre-built set of gestures that can be extended by using a gesture recorder application (IGT). These gestures can be evaluated to interact in the context of the ITS application by being the input of prototypes in ProtoActive.

4.1.2.3 Evaluate the system to detect flaws resulting from previous steps

The benefits of user-centered design for ITS applications rely in having users involved in the design process [2, 3 and 26]. ProtoActive helps designers involve users in early stages of the design process by taking advantage of the low-fidelity prototype' features of the tool, by being easy and fast to use, and by having prototypes with a *dirty* look, thus eliciting more user feedback as affirmed by Buxton [24].

4.1.3 Customizable interactions

A constant drawback among the studied prototyping tools was the lack of customization for users to interact with the prototypes during usability studies. Allowing designers to create custom gestures allows the creation of new ways to interact that might better suit for a certain task or group of users. ProtoActive provides a set of pre-built gestures that can be expanded through the use of IGT, a tool that allows designers to provide samples of a gesture to create new gesture definitions that can be used to interact with the prototypes. This feature was gathered from drawbacks from the following tools: *CrossWeaver* (section 3.2.1), *Balsamiq Mockups* (section 3.3.1), *Pencil* (section 3.3.2), *Fore UI* (section 3.3.7) and *Proto.io* (section 3.3.8).

4.1.4 No programming step required

The prototyping tools that allow custom interactions also come with the cost of requiring a programming step for customization. This was seen as a drawback as it adds to the cost of prototyping (more time or even the involvement of software developers to create the customization). In ProtoActive, no step requires any programming effort, allowing designers to fully create a prototype without requiring programming skills (also respecting the guideline explained in section 4.1.1.2). Creating prototype pages, linking them through gestures, creating custom gestures and evaluating them can be accomplished in ProtoActive through its GUI without any programming effort. This feature was gathered from drawbacks from the following tools: *Raptor* (section 3.2.2), *Sketchify* (section 3.2.3) and *Microsoft Sketch Flow* (section 3.3.6)

4.1.5 A prototyping tool should not constrain creativity

By allowing designers to sketch in a similar fashion as sketching on paper, ProtoActive allows designers to create interfaces that are not constrained by a pre-built set of controls. Among the tools studied, a constant problem was the lack of a feature that allows designers to free-hand sketch pages. Having pre-built UI widgets might increase the productivity and the speed of creating prototypes, but this comes at the cost of constraining creativity and especially for the design of ITS applications that is a field that is still evolving (and so are the UI widgets used in these applications). ProtoActive is a sketch-based prototyping tool that proposes to mimic the visual refinement of paper prototypes. This was done according to Buxton's principles [24] about sketching and low-fidelity prototypes looking quick and *dirty* which encourages users to provide more feedback. As a sketching tool, a reference in usability is paper prototyping. Having a sketch-based prototyping tool was gathered from drawbacks from *UISKEI* (section 3.2.4), *SILK* (section 3.2.5), *DEMAIS* (section 3.2.7), *Balsamiq Mockups* (section 3.3.1), *Pencil* (section 3.3.2), *iPlotz* (section 3.3.3), *AxureRp* (section 3.3.4), *MockingBird* (section 3.3.5), *Microsoft Sketch Flow* (section 3.3.6), *Fore UI* (section 3.3.7) and *Proto.io* (section 3.3.8).

There is however a way to also help designers in further steps of the design, by allowing them to import high-fidelity images of prototypes into ProtoActive and link these pages using ProtoActive's features. Using background images and linking them is better discussed in section 4.3.1.7.

4.2 Design guidelines based on interviews

A qualitative study was conducted in order to gather requirements for a prototyping tool of ITS applications. The author and a collaborator conducted semi-structured interviews with five UX designers from industry. The author contacted participants through *Calgary UX* mailing list, a user experience group based in Calgary [76] and the number of participants followed the recommendation from Nielsen *et al* [77]. The semi-structured interviews lasted around 40 minutes each and covered usability issues of sketching on a multi-touch device. The script used as a guide for the interview can be seen in section 9.10. The interviews aimed to collect the experience from the designers with other prototyping tools and get their opinion about the features that a sketch-based prototyping tool for touch-based applications should have. Besides the semi-structured interview, participants could use paper, tablets or tabletop devices that were available at the interview location, to demonstrate behavior and functionalities. The recorded audio of the interviews was transcribed and among with the notes taken during the interview and based on the partial structure of the interview, design guidelines were defined.

The following sub-sections discuss the requirements for ProtoActive based on these interviews. For explanation purposes, the requirements will be discussed in two groups: drag and drop controls and sketching tools.

4.2.1 Drag and drop controls

This issue was brought up during interviews by suggesting that ProtoActive could be a widget-based drag and drop tool. The feedback was that standard *WIMP* controls are not widely used in ITS applications therefore having a list or pre-build widgets would bias designers to use pre-built widget and thus limiting creativity. One of the participants commented: *"I've been working with*

3D applications (for ITS) *for a while and the concept of these components* (pre-built widgets), *they don't quite apply".*

In order to avoid the use of drag and drop widgets, ProtoActive was designed to be a free-hand sketching tool, without drag and drop widgets, any filtering or widget recognition of the drawings. Not recognizing widgets follows Buxton's [24] principle that low-fidelity prototypes should look *dirty* and that they are fast to create.

4.2.2 Sketching tools

The distinction between sketching and prototyping can be surpassed in ProtoActive as it gives designers the opportunity to create simplistic paper-like sketches that do not demand much time and effort. ProtoActive also allows designers to create prototypes that focus on usability (see Figure 2), with prototypes that allow the evaluation of design ideas as well as interaction. From the feedback of the qualitative study, a sketching tool should be the closest to a real sketching scenario with paper and pencil, which differentiates ProtoActive from a drawing tablet where the input (on the drawing tablet) is separate from the display space (screen display). In ProtoActive sketches are performed on touch-based screens, meaning that in a similar fashion as paper, the sketches are visualized on the same display where they are made. Regarding using drawing tablets, one of the participants commented: *"...I've tried to use a drawing tablet but realized that it was faster to sketch it and scan it".*

A sketching tool should be simple and fast to use, or else designers might just opt for traditional paper and pencil. In order to improve the sketching experience in ITS devices, participants were asked how drawing using ITS applications could be improved and three options were presented by the researcher and discussed with the participants:

- having the touch bounds determine if the designer wants to erase or draw (e.g. using the thumb vs. a fingertip). This option was well accepted but did not perform well due to misrecognition between drawing and erasing mode, in principle caused by hardware limitations;
- having a voice activated command that would change the drawing mode regarding to voice commands “*eraser*”, ”*selector*” and “*pen*”. This approach was not well accepted as it would be disturbing for other co-workers;
- user presses and holds a button with one hand to change the drawing mode from erasing, selecting or drawing; whilst the other hand performs the action on the sketching canvas.

The approach that received the best feedback was the third option. ProtoActive drawing features consist of:

- a canvas area that can be drawn using fingers or stylus pen;
- an eraser functionality;
- a selection button to select strokes on the canvas to move, resize or remove;
- a color button to change the color of the stroke;
- an undo button;
- a gesture area button that allows designers to draw an area on the canvas to define a *gesture area.*

A *gesture area* is an area defined in a prototype’s page by the designer that can contain a list of gesture and prototype’s page associations. When a designer defines a *gesture area* in a prototype, he can associate a gesture with this area by choosing from a list of pre-defined gestures or define a custom gesture. After selecting the gesture, the designer is asked to choose which page of the prototype he wants the prototype to navigate to when the gesture is recognized during a user study. To remove selected strokes or *gesture areas* from the canvas, a designer drags the selected strokes or *gesture area* to the right side of the canvas (to the *trash can* area), in a similar fashion as a designer would move or remove an object placed on top of a sheet of paper.

The following sections will explain ProtoActive in two aspects: as a tool and its features to design prototypes, and as a tool to evaluate prototypes.

4.3 ProtoActive features

With the requirements gathered from related work and interviews with UX designers, ProtoActive was created to allow designers to feel comfortable to create different flows and design concepts without much effort. Section 6.1 describes a study conducted to among other features, evaluate ProtoActive`s interface. The final design of the tool accommodating the requirements is shown in Figure 12. This section will explain ProtoActive's features.

4.3.1.1 Sketching canvas

ProtoActive was designed to keep the sketching area to a maximum (which can be seen in Figure 12 as the empty white space). Sketching in ProtoActive can be conducted with free-hand or with a stylus pen. No drag-and-drop or sketch recognition was implemented in ProtoActive and the only filter added to the sketching canvas is a "*FitToCurve*" feature that smooths out the stroke.

4.3.1.2 Define gesture areas

During usability studies, prototypes in ProtoActive can be interacted with via gestures and this can be done through the use of *gesture areas*. *Gesture areas* are user-defined areas in a prototype page that can be associated with one or many gestures that during evaluation, the recognition of these gestures will trigger the prototype to change to the specified page. The *gesture area* button

(Figure 12, item 1) allows designers to define gesture areas by performing a lasso on the screen, the bounding area of the lasso will become a *gesture area* in the prototype. Further explanation of gesture areas can be found in the following section 4.3.1.3 and section 4.4.

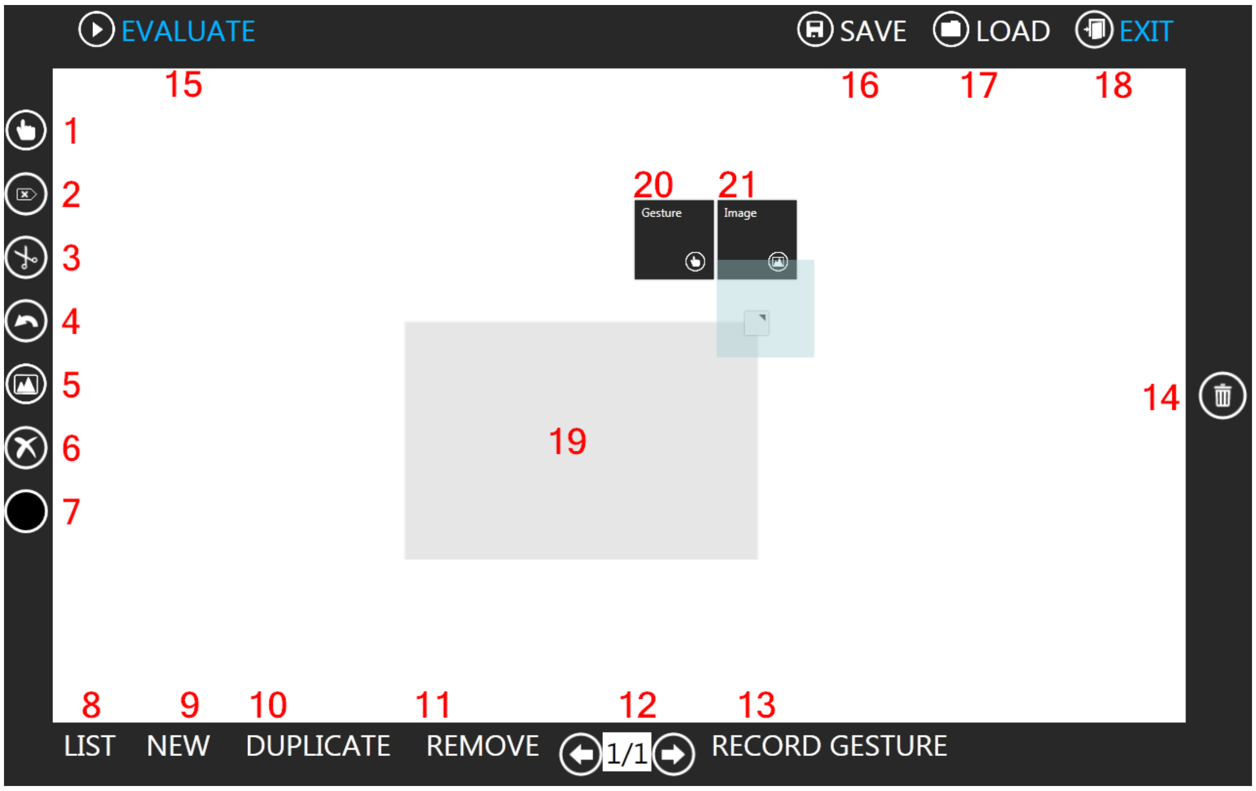


Figure 12 ProtoActive screenshot

4.3.1.3 Gesture area

Gesture areas (Figure 12, item 19) are movable and resizable areas on the page of a prototype that can be bound to one or multiple pairs of gesture and page. To do so, from a gesture area, the designer can select the *gesture menu* (Figure 12, item 20), which will bring up the gestures triggers dialog (Figure 13) where a designer can use a pre-built set of common gestures and bind its detection to showing a specific page on the prototype chosen by the designer:

- *Tap*, a single tap with the finger on the surface;

- *Double Tap*, subsequent taps with the finger on the surface;
- *Pinch*, gesture using two fingers moving towards each other;
- *Swipe left*, single finger moving left;
- *Swipe right*, single finger moving right;
- *Lasso*, single finger gesture of an arbitrary shape establishing a closed loop;
- *Zoom*, gesture using two fingers moving in opposite directions.

If a designer wants to use a gesture that is not listed, he can create custom gestures using IGT by clicking on *Add custom gesture* button (Figure 13) or *Record Gesture* (Figure 12, item 13). ProtoActive is integrated with IGT, allowing the designer to create custom gestures and evaluate them with the prototypes. Any custom gesture created in ProtoActive through IGT will be automatically available for all the projects on the *gesture triggers* list (Figure 13). The process to create custom gestures will be explained in Chapter Five. Another requirement that came from the qualitative study shows that it would be useful to allow designers to fill the *gesture area* with images. By selecting the *image* menu from the *gesture area* (Figure 12, item 21) a designer can select an image from his computer.

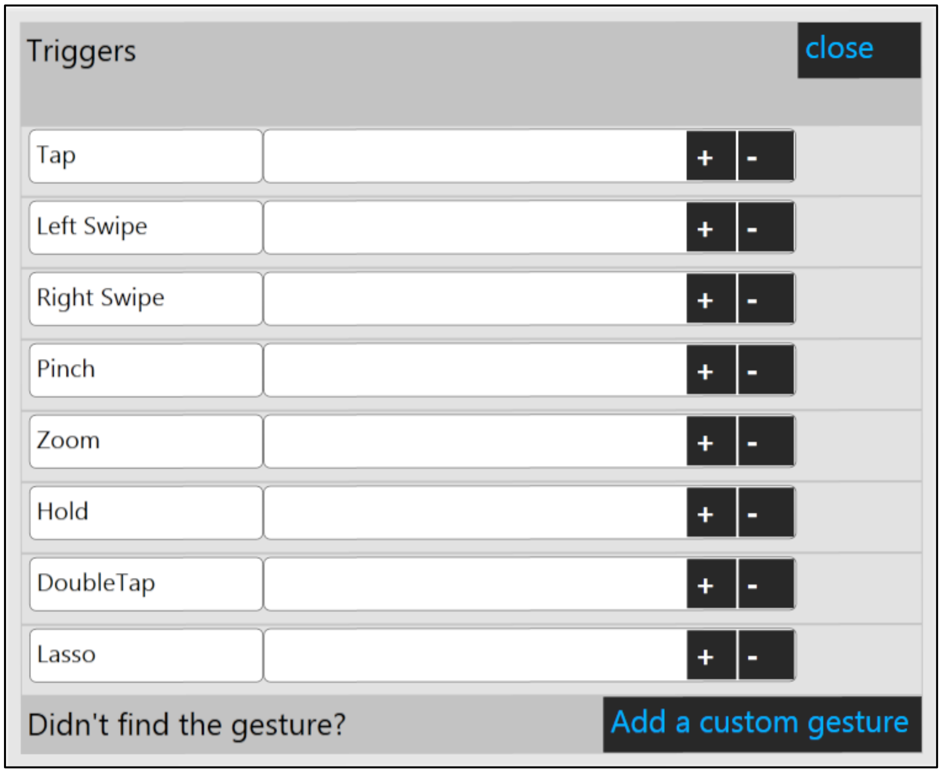


Figure 13 Gesture area trigger selection

4.3.1.4 Eraser

An eraser button (Figure 12, item 2) that when selected allows the designer to erase strokes using his finger.

4.3.1.5 Selecting strokes

To help designers simulate movement and zoom features in their applications, ProtoActive allows stokes to be selected, resized and moved on the canvas. By clicking on the scissors button (Figure 12, item 3) a designer can perform a lasso on the canvas to select all the strokes inside the lasso. With the selected strokes, a designer can move them on the canvas, remove them (by dragging them to the *trash can* area) or resize them (by using zoom or pinch gesture).

4.3.1.6 Undo strokes

The *undo* button (Figure 12, item 4) allows designers to undo stroke mistakes. Removing pages, *gesture areas* or selected strokes cannot be undone.

4.3.1.7 Adding and removing background images

ProtoActive also allows designers to set the background of a page, by the two buttons: *add background* (Figure 12, item 5) and *remove background* (Figure 12, item 6). This was required in order to create prototypes that work with a static image as a background. An example of the value of this feature is shown by Unger and Chandler in their book UX Design [39]. In the prototype chapter of the book, the authors explains how a designer can create prototypes in a *What You See Is What You Get* (WYSIWYG) tool such as Dreamweaver CS4 [90] and export the prototype pages as separate images and set them as a clickable background in an HTML page. A similar solution can be achieved with ProtoActive by using the *set background* (Figure 12, item 5) feature to set the background of a prototype page with images from other applications. In ProtoActive, after setting the background of a page a designer can still draw on the top of the image and add *gesture areas,* which allow setting specific parts of the background to respond to gestures and trigger page transition.

4.3.1.8 Stroke color button

In order to allow the design of colored prototypes (allowing designers to highlight some areas with a specific color) the *color* button (Figure 12, item 7) allows the designer to select the color of the stroke on the canvas.

4.3.1.9 Page list and navigation buttons

This feature allows designers to easily navigate through the prototype's page. *List* (Figure 12, item 8) shows a list of thumbnails of all the pages in the prototype. The *navigation* buttons (Figure 12, item 12) changes the page to the previous (if there is any) or to the next page (if there is any).

4.3.1.10 New, duplicate and remove buttons

ProtoActive allows designers to create prototypes with multiple pages. The *new* button (Figure 12, item 9) creates empty pages on the prototype, while *duplicate* button (Figure 12, item 10) creates a duplicate of the current page in the prototype with the same drawings, and *gesture areas* of the original. It allows designers to have modified versions of the same page without much effort. The *remove* button (Figure 12, item 11) will remove the current page of the prototype.

4.3.1.11 Trash can area

To facilitate the removal of gesture areas and strokes, the right side of ProtoActive has a *trash can* area (Figure 12, item 14) where *gesture areas* and selected strokes can be dropped and removed from the canvas.

4.3.1.12 Run evaluation

The *evaluate* button (Figure 12, item 15) switches ProtoActive into *evaluation mode*. In *evaluation mode*, the canvas turns full screen, non-editable and the pages can be navigated through the detection of the gestures defined in the *gesture areas*. This navigation flow will be explained in further detail in section 4.4.

4.3.1.13 Saving, loading and exiting a project

The *save* (Figure 12, item 16) and *load* (Figure 12, item 17) buttons allow designers to export they designs to different devices running ProtoActive. Loading a ProtoActive project created from a different devices works normally, but the gestures detected will be limited by the capability of the device (e.g.: detection of hand only works in Microsoft Surface, having a prototype that uses hand gestures in another device will work without reacting to hand gestures). Finally the *exit* button (Figure 12, item 18) closes the application.

4.4 User studies with ProtoActive

The main purpose of a prototyping tool is to elicit user feedback about design ideas; so in order to allow designers to evaluate their prototypes, ProtoActive has an *evaluation mode* (section 4.3.1.12) as shown in Figure 14 , in where all the controls from ProtoActive disappear, showing only a "*leave*" button that turns ProtoActive back into designing mode. In *evaluation mode,* the only way to move through pages is through the *gesture areas* that now are invisible, unless they have a background image (for details on this, see section 4.3.1.3). Each *gesture area* can have one or many gesture-page bindings. If a gesture from the gesture-page binding is detected on the *gesture area*, the prototype will show the corresponding page of the binding.

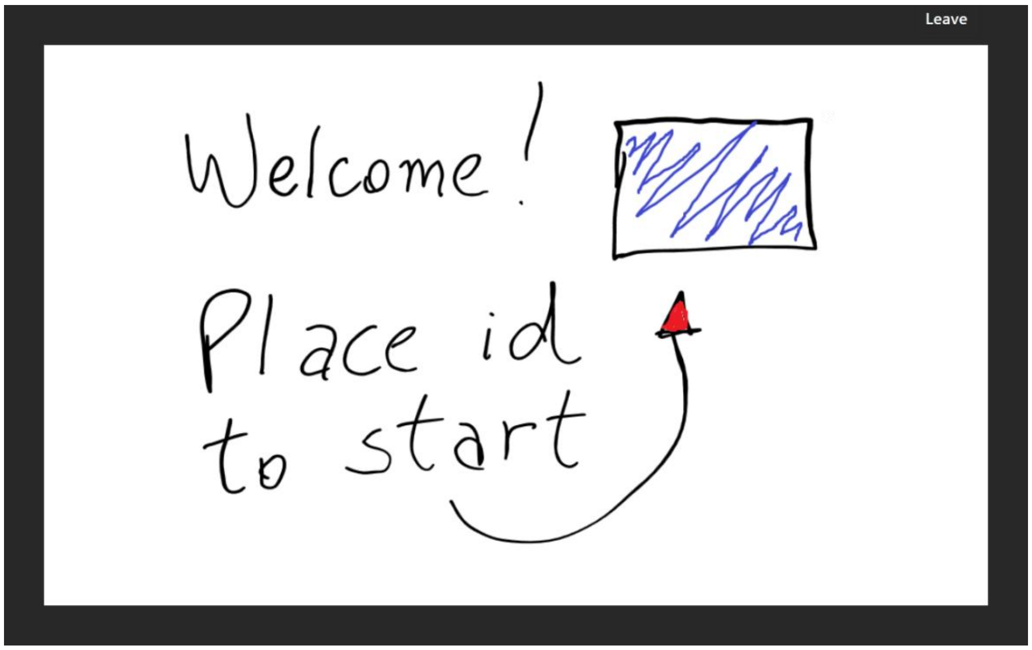


Figure 14 ProtoActive in *evaluation mode*

Figure 15 illustrates how a designer can use a sequence of pages to simulate the behavior of the application. When evaluating the prototype in Figure 15, the first screen (quadrant 1) is a login screen that has a *gesture area* that is activated when a certain tag is placed over the gesture area. Quadrant 2 shows a scan image screen with images A, B and C. Image B has a *gesture area* that is bound to two gestures:

- *"X"* gesture, that navigates to quadrant 3, meaning that image B was deleted;
- place *open right hand* gesture that navigates to quadrant 4, where it shows image B selected and with a menu.

Both *open right hand* detection and *"X"* gesture can be created using IGT, thus making these two gestures available in ProtoActive. Having the newly created gesture available, the designer can bind it to a *gesture area* and use it in the low fidelity prototype to trigger a page transition as seen in Figure 15. It is important to mention that all the binding can be done without writing one single line of code.

In *evaluation mode* ProtoActive was designed to be used by one user due to its successive state in page transition. When a gesture from a gesture/page pair is detected by a *gesture area* in *evaluation mode*, ProtoActive instantaneously changes the active page to the page from the gesture/page pair of the *gesture area.* Another limitation decided as a design guideline of ProtoActive is the absence of animations between transitions. ProtoActive was designed to simulate the experience of prototyping using paper; with this guideline, animations can only be mimicked as page transitions.

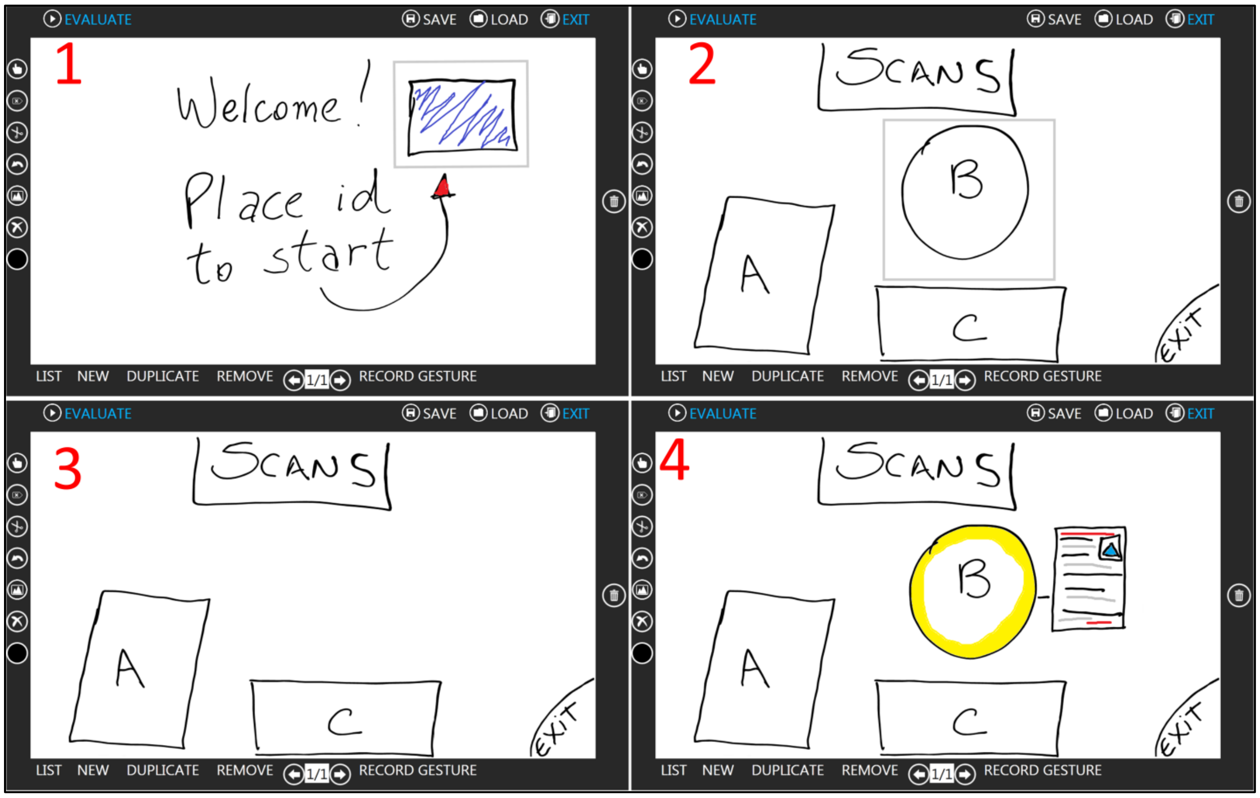


Figure 15 Using gestures to navigate between the pages of the prototype in ProtoActive

The process to create gestures will be described in Chapter Five, which will describe IGT, the requirements of a gesture learning tool, its usage flow and how it integrates with ProtoActive. Figure 14 shows the page in quadrant 1 from Figure 15 in evaluation mode. Having *evaluation mode* starting from the current page instead of the first page allows designers to evaluate a specific part of a bigger prototype without having to navigate until the desired page in *evaluation mode*.

The authors asked participants how they currently collect user feedback when evaluating their prototypes. The participants responded that they record the audio of the evaluation sessions using personal recording devices and add notes about some of the user's comments. For video recording, participants commented that aspects like posture or ergonomics are important and should be taken in consideration when evaluating a prototype, especially for an ITS application where size constraints and portability (for tablets) impact the user experience, meaning that a screen recording feature would not cover the needs and would most likely not be used. These constraints were considered out of the scope for ProtoActive as it was brought up by the designers that they would rather use their portable equipment to record as it is what users are more comfortable with. This can be considered as future work to this research and will be discussed in section 7.5. Another way to gather feedback from users in ProtoActive is to allow users to draw suggestions on the prototype pages during user evaluations in ProtoActive.

CHAPTER FIVE : GESTURE LEARNING IN PROTOACTIVE WITH IGT

"*As technology becomes more and more pervasive, it is finding itself in increasingly diverse and specialized contexts. The technologies that we design do not, and never will, exist in a vacuum. In any meaningful sense, they only have meaning, or relevance, in a social and physical context*" [24 p.32].

Based on the study by Morris *et al* [35] that suggests that having user-centered design of gestures will produce a gesture set that will better suit users' needs, ProtoActive comes with a gesture recording tool (IGT) that allows designers to create gestures without having to deal with the cost and difficulty of programming by utilizing a machine-learning algorithm that uses samples provided by the designer to learn a gesture definition [68]. After, ProtoActive allows the evaluation of the gestures in user-centered studies with prototypes that can be interacted with the custom gestures. The author proposes that this solution will allow designers to explore more and different interactions by not having to deal with the cost and difficulty of creating custom gestures through programming code, but keeping a user-centered design approach.

IGT is based on the *Gesture Toolkit*, a tool developed by Khandkar *et al.* [10]. To help understand IGT, some concepts of *Gesture Toolkit* and how IGT extends *Gesture Toolkit* will be explained in the following section.

5.1 Requirements for IGT

Gesture Toolkit simplifies the process of gesture recognition and definition by providing software developers with a toolkit that contains a domain-specific language that helps the process of defining new gestures and allows it to be used across multiple hardware platforms [10]. *Gesture Toolkit* recognition framework architecture is shown in Figure 16.

The process of gesture recognition and definition in *Gesture Toolkit* is done through writing code and therefore it is target to be used by software developers implying in costs to generate a prototype that can be used and evaluated by the user of the application.

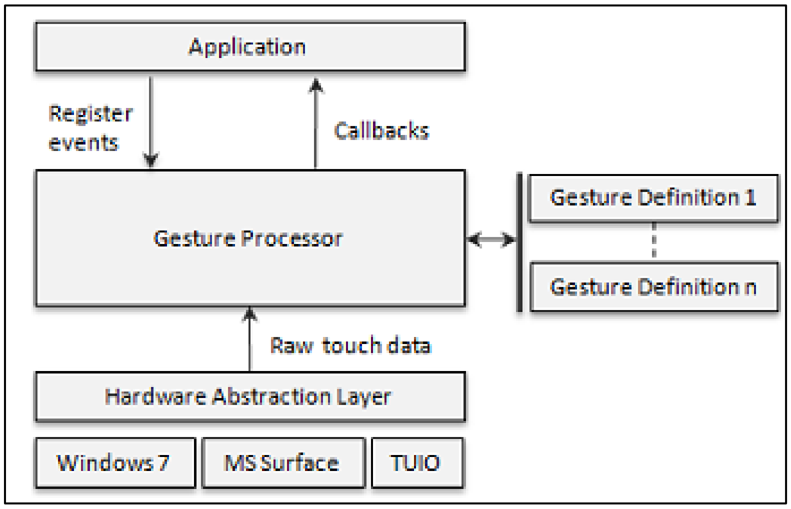


Figure 16 The architecture of gesture recognition engine in *Gesture Toolkit*, extracted from [10]

This thesis aims to modify *Gesture Toolkit* by removing the need for software development expertise to use it and through this also allow:

- easy definition of gestures based on provided samples;
- designers to provide samples and try them on a provided interface;
- the definition of gestures containing fiduciary markers and hand detection;
- a gesture definition that can cover pre-defined nuances in gestures.

These motivations served as a requirement to build an interface that allows designers with no software development expertise to define gestures (including multi-touch, detection of hand and fiduciary markers) by providing samples of the gesture they want to define and this definition accomplishes all the nuances between the provided samples. The modification to *Gesture Toolkit* and the interface to provide samples and define gestures was named *Intelligent Gesture Toolkit*

(IGT). Figure 17 shows the architecture of IGT and how it extends *Gesture Toolkit*. The extensions will be explained in the following sections:

- the interface where designers will provide the samples, manage the samples in a repository and try them;
- the *touch analyzer* that analyzes the samples in order to improve the accuracy of the gesture definitions;
- the specific modifications to GDL;
- and the *anti-unification* algorithm that receives the sample of the gestures filtered by the *touch analyzer* and outputs a gesture definition. Figure 17 also shows that the gesture definition produced by IGT can be used in other applications using IGT as a gesture recognition framework, such as ProtoActive.

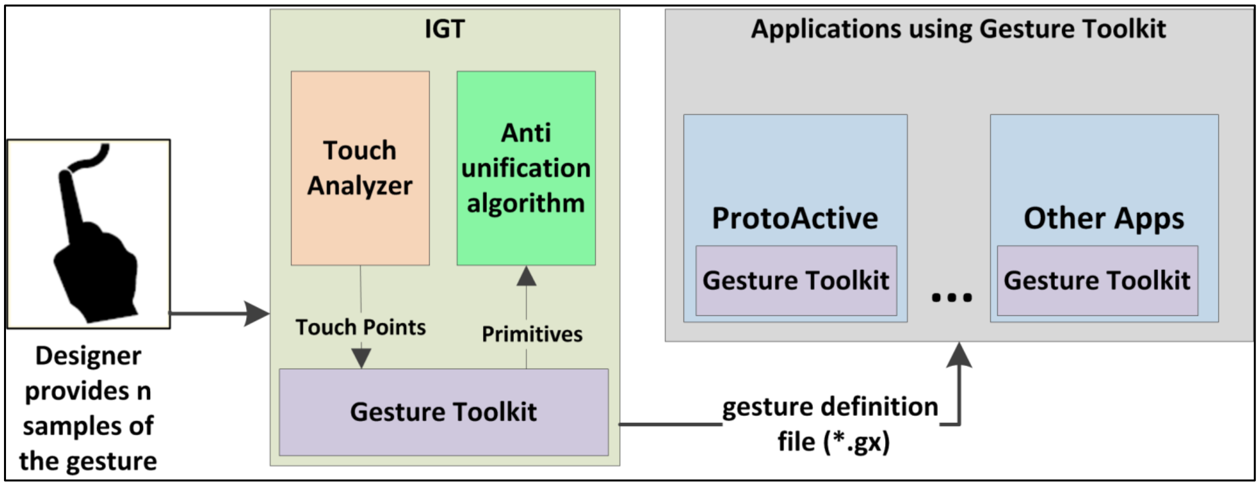

IGT
Applications using Gesture Toolkit
Touch Analyzer
Anti unification algorithm
ProtoActive
Other Apps
Gesture Toolkit
Gesture Toolkit
...
Designer provides n samples of the gesture
Touch Points
Primitives
Gesture Toolkit
gesture definition file (*.gx)

Figure 17 IGT Architecture

5.2 Recording gestures in IGT

A gesture definition needs to be as broad as necessary to surpass the nuances of different users performing the gesture in different moments. A gesture also needs to be as precise as possible to avoid conflict with other gestures and to be detected only when this gesture is really intended by the user. In order to gather the terms and the different nuances to define a gesture, IGT asks the designer to train the tool by performing samples of the gesture they want to create. It is up to the designer to provide samples that cover all the nuances they desire the gesture definition to cover. It is also up to the designer to create the gestures providing the samples himself or by asking users to provide samples to generate gesture definitions as based on the study conducted by Wobbrock *et al* [6] that compared gesture sets created by HCI experts to gesture sets created by users.

Based on experimentation and custom heuristics, a sample is considered outside the standard when less than 20% of its primitives match any of the previous samples. The designer can choose to keep the non-standard sample, which creates a more general gesture, or the designer can remove the sample and add another one. IGT does not require a previously defined training set in order to be calibrated or to function properly. However, there is an established moment when the designer uses the tool to define the gesture. As an example, we will show how a designer can define a gesture that consists of a hand placed on a surface and a finger performing a vertical line as shown in Figure 18.

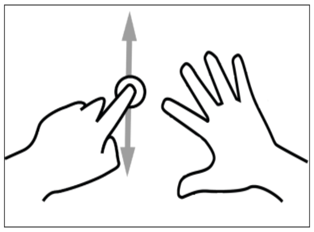


Figure 18 Vertical finger and hand gesture

By clicking the *“Record gesture”* button as seen in Figure 19, the designer has three seconds to perform the “v*ertical line and hand”* gesture on the canvas. Having a time for the designer to perform a gesture, allows the creation of gestures that can be detected by holding a position in contrast to only detecting a gesture on the event of releasing the touch. By performing the gesture on the canvas, IGT produces a gesture definition in two steps. The first one defines a right hand placed open on the tabletop while the second step defines a downward line performed with a finger making a path of 100 mm; finally a relation between the two is established, stating that the first step is placed to the right of the second step.

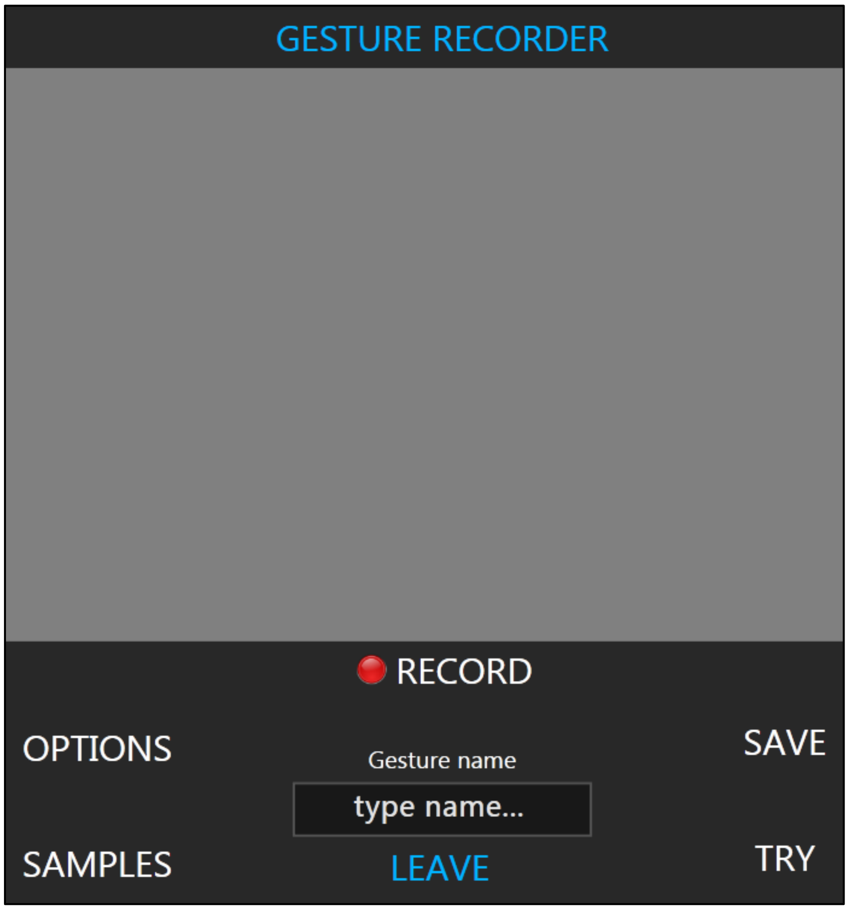
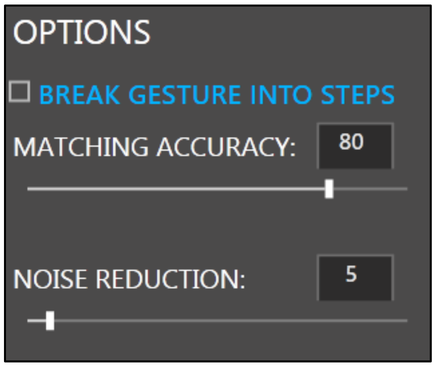


Figure 19 IGT screenshot

Using GDL to define gestures allows gesture definitions that are readable (as shown in Figure 35); the designer can read the definition of the sample in the sample repository dialog as seen in Figure 21. If the designer does not agree with the gesture definition shown in Figure 21, he can remove the gesture definition and submit a new sample. If the designer finds the definition too specific, they can perform the gesture a second time, which will generate a new definition. For this example, consider that the second sample was performed in a similar fashion to the first sample but with an upward line making a path of 400 mm.


OPTIONS

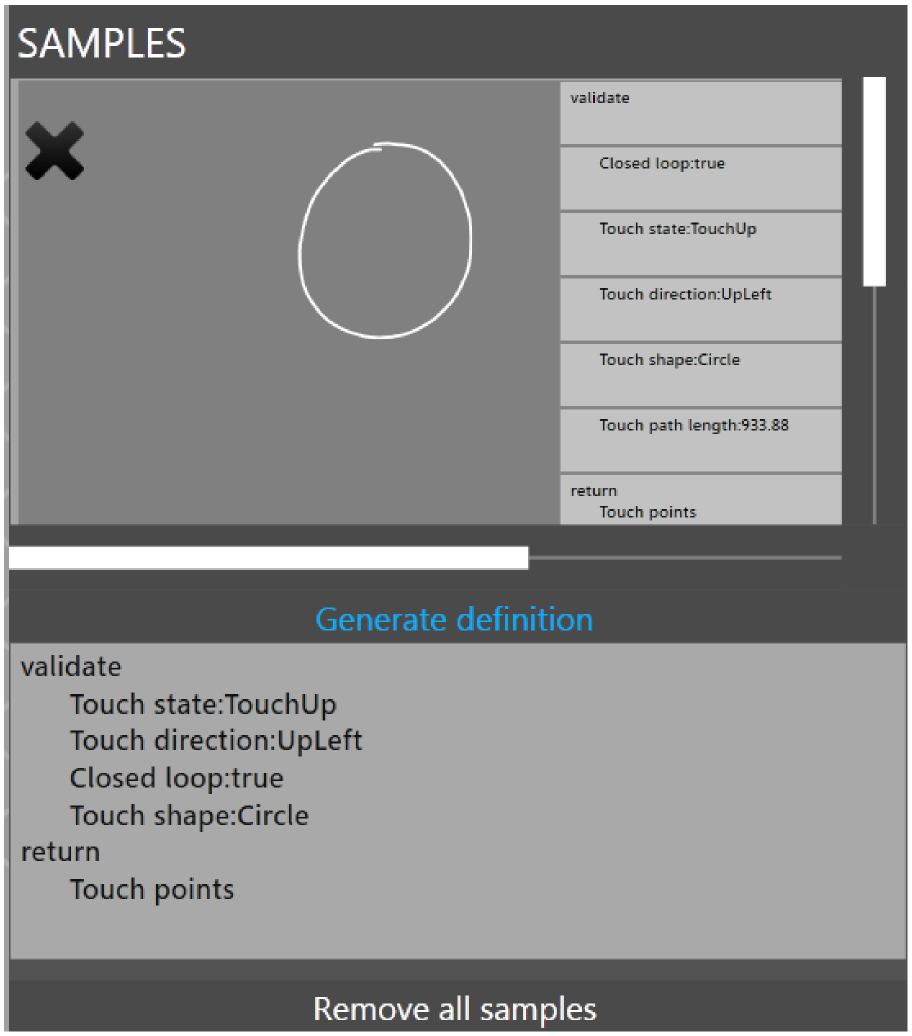
☐ BREAK GESTURE INTO STEPS

MATCHING ACCURACY: 80

NOISE REDUCTION: 5

Figure 20 IGT option dialog

By clicking the "*Generate definition*" button, IGT creates a gesture definition that would be recognized both samples.



```
validate
    Closed loop:true
    Touch state:TouchUp
    Touch direction:UpLeft
    Touch shape:Circle
    Touch path length:933.88
return
    Touch points
```

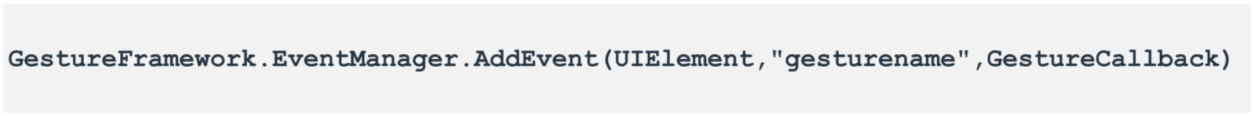
```
validate
    Touch state:TouchUp
    Touch direction:UpLeft
    Closed loop:true
    Touch shape:Circle
return
    Touch points
```

Figure 21 IGT Sample repository

In order to create a gesture definition that can recognize. The created gesture definition covers the first and second sample, detecting a hand placed on the surface and a finger making a vertical line that can vary from 100 mm to 400 mm. If this definition is still too specific, the designer has two options:

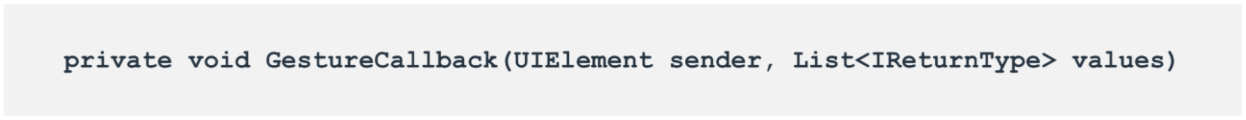
- provide more samples to generate more variances for the anti-unification algorithm
- change "*MATCHING ACCURACY*" value in options as seen in Figure 20, to a lower value.

When the designer agrees with the gesture definition provided, he can try the gesture in IGT by clicking "*TRY*" button in the IGT (Figure 19). If the designer is satisfied with the gesture recognition, he can save the gesture thus making it available in a *.gx* file which can be read and used by any application using *Gesture Toolkit* with IGT extension to GDL to detect gestures. In order to allow newly created gestures in IGT to be used in other applications without having to re-compile the applications, *Gesture Toolkit* was also modified with a method that searches in the folder *"/Resources/Custom"* inside the application folder for new ".gx" files that contain gesture definitions. The found gestures become available to the application and just need to be bound to a user interface element as shown in Figure 22 and to a gesture callback method with a signature as shown in Figure 23.



```
GestureFramework.EventManager.AddEvent(UIElement,"gesturename",GestureCallback)
```

Figure 22 Gesture and UI element binding method



```
private void GestureCallback(UIElement sender, List<IReturnType> values)
```

Figure 23 Signature of a gesture callback method

As ProtoActive is a tool designed to be used by designers with no software development experience, the binding of a gesture to an interface is done by the application without requiring the designer to write any code. This binding process is made in ProtoActive through the use of *gesture areas* as explained in sections 4.3.1.2, 4.3.1.3 and 4.4.

```
name

/* name of the gesture*/

validate

/* validate block that can consist of one or more primitives and one or more steps*/


return

 /* What will be returned when the gesture is detected*/
```

Figure 24 Skeleton of a gesture definition in GDL

5.3 Gesture Definition Language

A gesture definition in *Gesture Toolkit* has a name that uniquely identifies a definition within the application; one or more validation blocks that contain combinations of primitive conditions and a return block that contains one or more return types. In *Gesture Toolkit* a gesture is defined in a domain-specific language named *Gesture Definition Language* GDL [10]. A gesture definition in GDL consists of a name, a validation section and a return section as shown in Figure 24. GDL provides a set of primitive conditions to define the gestures. To allow a wider range of gestures and also to better fit the needs of the anti-unification process, it was necessary to add new features to GDL: the ability to have a relationship between primitives; primitives that express the value of tags and finally a primitive that specifies hand postures.

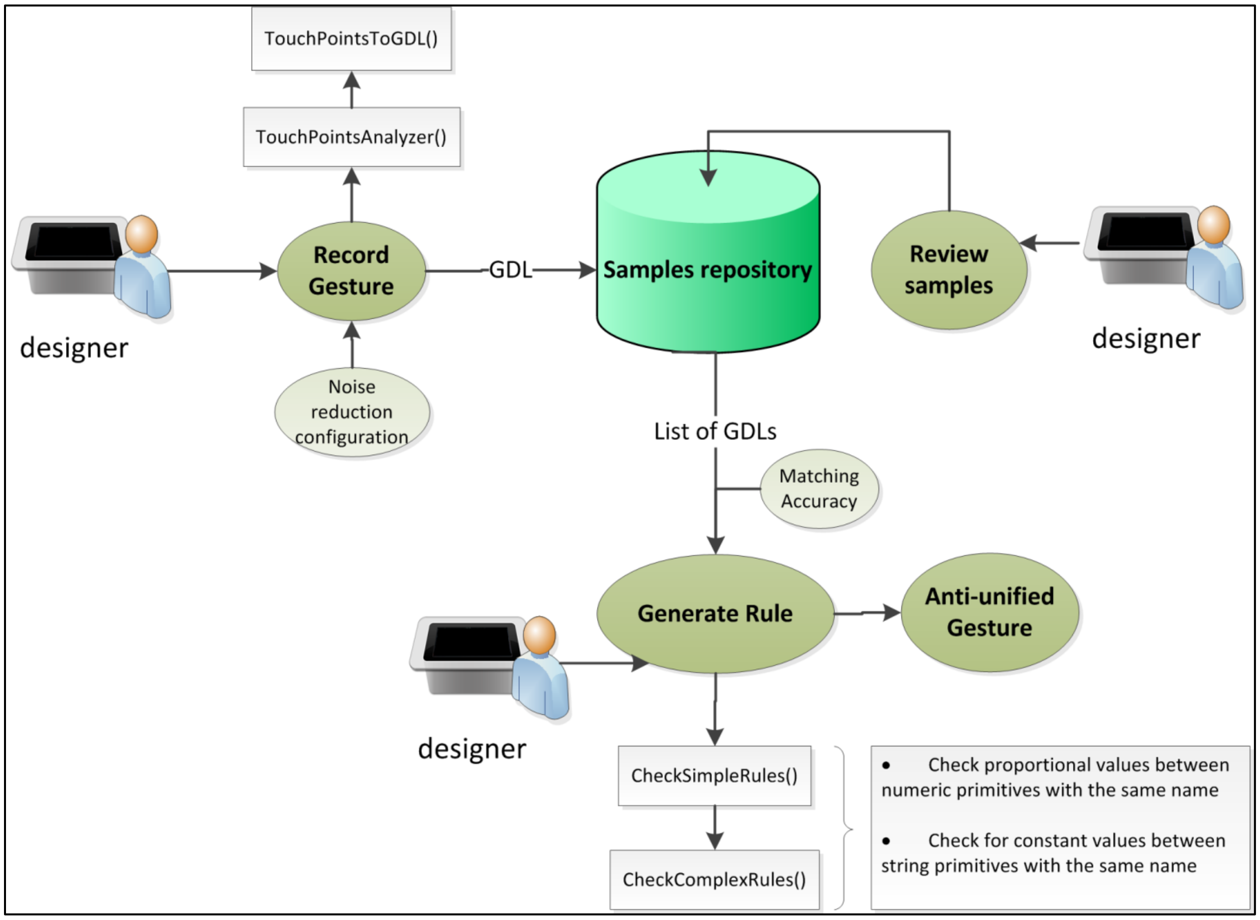


Figure 25 IGT diagram of an anti-unified gesture definition

Figure 25 contains a diagram of the functionality flow in IGT: a designer recording samples of a gesture on IGT and then based on the samples; create an anti-unified gesture. The designer records samples of a gesture that among with the noise reduction parameter in the options dialog (Figure 19) is analyzed by the *touch analyzer* (for further details see section 5.7). The method *TouchPointToPrimitive()* (pseudo-code of the method can be seen in section 9.3) analyzes the touch points in a sample and validates the touch points based on the primitives defined in *Gesture Toolkit* (a list with all the primitives to date can be found in the extended GDL grammar in section 9.4). The list of validated primitives is presented to the designer in GDL. The GDL of the recorded sample is stored in a sample repository that can be reviewed and removed in the samples dialog (Figure 19) by the designer. Based on the recorded samples in the repository, the designer can review the samples and either decide to generate an anti-unified gesture definition based on the samples (this will be explained with further details in section 5.9) or remove undesirable samples based on the GDL of the samples. After generating the anti-unified gesture definition, the designer can perform the gesture on IGT and evaluate its recognition. If the designer is not satisfied, he can provide more samples or remove undesired samples. When providing samples for a gesture definition in IGT, the touch points from the sample will be filtered and then analyzed through a matching process with the pre-built primitives in *Gesture Toolkit.* The matching process between the touch points of a sample and the primitives works in four steps: match touch points into pre-defined shapes, add properties to shapes, identify relationships between shapes and establish temporal and event values.

5.3.1 Match the touch points into pre-defined shapes

These primitives attempt to match the provided touch points with pre-defined shapes based on arithmetic calculations for each shape.

- *Tag:* detection of a tag placed on the device, this will be seen in more detail in section 5.8.2;
- *Hand:* detection of a hand placed on the device, this will be seen in more detail in section 5.8.3;
- *Circle:* single finger movement where the continuous trajectory of the stroke defines a circle;
- *Box:* single finger movement where the continuous trajectory of the stroke defines a quadrilateral figure;

- *Line:* single finger movement where the continuous trajectory of the stroke defines a line, this will be seen in more detail in section 5.8.1 ;

5.3.2 Add properties to these shapes:

For the identified shapes, *Gesture Toolkit* adds values to the primitives that require a value.

- *Length:* calculates the length of a continuous stroke;
- *Tag value:* on tag detection, gets the value associated with the fiduciary marker;
- *Hand posture:* identifies the posture of a detected hand, explained with further detail in section 5.6;

5.3.3 Identify relationships between these shapes

These primitives identify if there is any relationship between the previously identified primitives.

- *Angle Between:* when detecting lines, informs the angle that two lines do between them;
- *Relative Position:* with more than one stroke, identifies the relative position between them. It can be *Top*, *Bottom*, *Left* or *Right*.

5.3.4 Establish temporal and events values

These primitives specify temporal, directional and event based aspects of a gesture.

- *Number of touch points:* Number of touch points on the device at the same time;
- *Touch actions:* when a step or whole gesture should be detected. It can be on touch detection, on touch movement or when removing the touch point (*e.g.:* moving up finger);
- *Touch time:* informs the duration of a gesture;
- *Direction:* calculates the direction of a stroke based on the eight orthogonal directions;

- *Order of the events:* defines the order in where the primitives are detected, it is expressed in form of steps of a gesture.

5.4 Relational Primitives

In order to express relationships between primitives in GDL, the values of the primitives were extended to accept values represented as variables. As a variable, a primitive can be proportional to another primitive in the gesture definition.

```
validate as step 1
        Touch state: TouchMove
        Touch shape: Line
        Touch direction: Down
        Touch path length: x
validate as step 2
        Touch state: TouchMove
        Touch shape: Line
        Touch direction: Right
        Touch path length: 1.5x ..2x
validate as step 3
        Touch state: TouchMove
        Touch shape: Line
        Touch direction: Up
        Touch path length: x
validate as step 4
        Touch state: TouchUp
        Touch shape: Line
        Touch direction: Right
        Touch path length: 1.5x ..2x
validate
        Touch limit: 1
        Angle between step1 and step2: y
        Angle between step2 and step3: 1.1y ..1.2y
        Angle between step3 and step4: 1.1y ..1.2y
        Angle between step4 and step1: 1.1y ..1.2y
```

Figure 26 Rectangle gesture defined using relational primitives in IGT

This for example, allows two primitives to have their values expressed as a relation of the other. E.g.: primitive 1 is the double of primitive 2. By finding a relationship between steps of a gesture, a gesture definition can for example, represent a *rectangle* gesture that consists of 2 lines of nearly equal length and two other lines that are equally proportional to the first two. Figure 26 illustrates a definition of the *rectangle* gesture. The values of the lines are all a multiplications of the value of the line found in step 1. The same occurs with *Angle Between*, where the subsequent values are dependent on the first value in *AngleBetween*.

5.5 Tag Recognition

This feature of IGT recognizes fiduciary tags on the tabletop allowing the definition of gestures that use such tagged objects. This feature allows designers to recognize any tangible object that contains a fiduciary tag attached to it. A designer can attach fiduciary tags on physical objects and have the prototypes detect when they are placed on the device or when they are lifted up from the device. By placing two fiduciary tags as shown in Figure 27, a designer can create separate gesture definitions that identify the position of the device. For example, in Figure 27, considering that the tag on the left is tag 1 and the other is tag 2, a designer can create a gesture definition where tag 1 is on the left of tag 2.

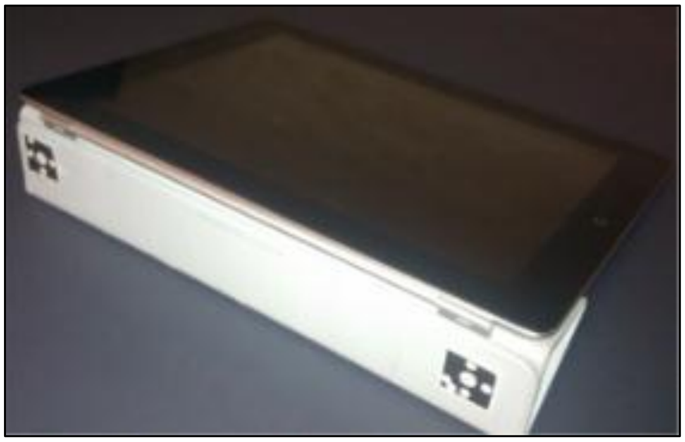


Figure 27 iPad with fiduciary tags

Similarly, another gesture definition can be created where tag 2 is on the left of tag 1. This will allow ProtoActive to determine the position that a tablet was placed on the prototype.

```
validate as step 1
        Touch tag: 5
        Touch state: TouchMove
validate as step 2
        Touch shape: 6
        Touch state: TouchMove
validate
        Relative position between step1 and step2: Left
```

Figure 28 Defining tag based gestures

5.6 Hand Recognition

To take advantage of devices that allow the recognition of *Binary Large Objects* (BLOB), IGT also allows the recognition of a set of hand postures allowing the definition of different postures of the hand.

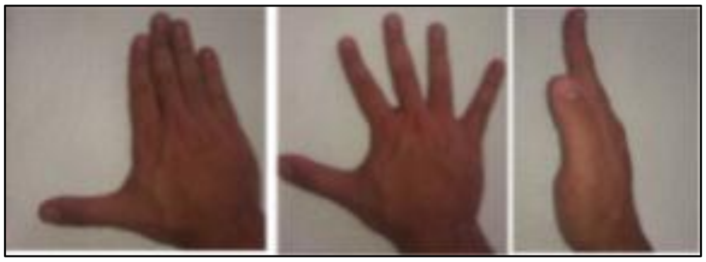


Figure 29 Hand postures recognized by IGT

The hand postures recognized are shown in Figure 29, from left to right: *Open*, *Spread* and *Vertical.* The orientation of the hand is also detected and set as a primitive's value: *Right* or *Left.* Figure 30 defines when a right hand with a vertical posture is moved up from the device.

```
validate
        Touch blob: Hand, Vertical, Right
        Touch state: TouchUp
```

Figure 30 Definition of a hand posture gesture

5.7 Gestures not recognized

For gestures that consist of touch points that cannot be matched in the pre-defined shapes or cannot be divided in steps of pre-defined shapes, the gesture recognizer will create a gesture definition that will not cover the nuances of the gesture and will lead to a gesture definition that will be more general and not specific to the intended gesture.

5.8 Touch analyzer

In IGT the sequence of touch points from the samples are pre-analyzed and matched with primitives by the *touch analyzer*. The architecture of the application is shown in Figure 17, which illustrates the *touch analyzer* and the anti-unification algorithm in the process. The *touch analyzer* is a functionality in IGT that analyzes the touch points of the sample of a gesture; it filters noise touch points in hand and tag detection and detects lines in a sequence of touch points. These analyses will be explained in the next sub-sections.

5.8.1 Stroke Analysis

A stroke is a continuous 2D sequence of touch points that ends when the finger moves up from the surface. In *Gesture Toolkit*, identifying steps of a gesture, helps to create a more accurate definition as it allows the identification of relationships between steps (see section 5.3.3). For example, Figure 22 shows a gesture definition of a “plus sign” gesture:

```
validate as step 1
        Touch state: TouchUp
        Touch shape: Line
        Touch direction: Down
        Touch path length: 290.3
validate as step 1
        Touch state: TouchUp
        Touch shape: Line
        Touch direction: Right
        Touch path length: 313.2
validate
        Touch limit: 1
        Angle between step1 and step2: 95.43
```

Figure 31 Gesture definition of a plus sign

Having pre-determined steps help IGT identify relationships between the steps (e.g.: *Angle between*), leading to a more accurate gesture definition. In the "*plus sign"* gesture example, this was possible due to the steps being determined by the designer lifting up his finger from the device when performing a line. In a continuous stroke, identifying parts of the stroke in lines can help identifying relationships between the lines. For example, in IGT, a *"check"* gesture definition generated without using the stroke analysis:

```
validate
        Touch state: TouchUp
        Touch direction: UpRight
        Touch path length: 353.4
```

Figure 32 Check gesture defined without stroke analysis

The problem with this gesture definition is that it is too general in terms of the shape of the gesture, leading to false positives in gesture recognition. A similar gesture generated with stroke analysis:

```
validate as step 1
        Touch state: TouchMove
        Touch shape: Line
        Touch direction: DownRight
        Touch path length: 105.3
validate as step 1
        Touch state: TouchUp
        Touch shape: Line
        Touch direction: TopRight
        Touch path length: 416.2
validate
        Touch limit: 1
        Angle between step1 and step2: 91.21
```

Figure 33 Check gesture using stroke analysis

This gesture definition is more specific to the gesture used as a sample and will lead to less false positives in recognition. The stroke analysis can be switched on or off through the "*Break gesture into steps*" checkbox in the options box (Figure 20).

When a designer provides a sample of a gesture, the *touch analyzer* filters the touch points of the sample and reduces the noise by removing unnecessary redundant touch points (touch points in the same location). For stroke analysis, a stroke is searched according to the *FindLines* algorithm (pseudo-code can be found in section 9.1) and continuous touch points can be identified as a line. The strokes are divided based on the direction of the stroke regarding the orientation of the application on the device. For example, by making a stroke in an "*L*" shape, the designer can make it with a downward touch and then moving to the right of the screen. The *Stroke analyzer* will break the stroke in two: one containing the touch points of the stroke that go down and the other containing the touch points of the stroke that go right.

The sensitivity of the analysis is set through the "*Noise reduction*" value in the option box in IGT. The noise reduction value is calculated as a percentage of the length of the stroke to determine the minimum size of a change in the direction of the stroke. A pseudo-code with the algorithm used for the stroke analysis is shown in section 9.1.

5.8.2 Tag Analysis

In order to allow the definition of tag-based gestures, the *touch analyzer* needs to remove noise touch points from the samples. This will allow IGT to create a more concise gesture definition. For tag recognition, noise touch points are defined as touch points that are detected from the placement of the tag on the device (e.g.: detecting parts of an object before detecting the tag of the object).

In order to filter noise touch points, the *touch analyzer* identifies whether there is a tag recognized in the touch points. If there is, all the points that are within the tag bounds are removed. For example, when placing a tangible object that contains a tag, some other parts of the tangible might touch the device and be recognized as touch points, the *touch analyzer* will remove these touch points. This filter only needs to be applied in the learning phase as for the recognition phase, the intermediate touch points that do not compose any registered gesture are not considered. Section 9.2 shows the algorithm used to remove noise touch points for tag detection.

5.8.3 Hand Analysis

Hand recognition is divided in two moments; the first one deals with acquiring images from the device on the event of a blob detection and the second deals with treating the acquired images to identify hand postures during gesture recognition (for pseudo-code, see section 9.8 and 9.9) and it is executed during gesture recognition.

5.8.3.1 Acquiring snapshots for hand recognition

In order to analyze specific hand details, the frames of the touch events are captured in a rate of 10 frames per second, using a custom heuristic based on results that aimed to get the best performance and the best recognition rate. In a similar fashion as Chang *et al.* [78] the hand analysis process can be divided in three steps:

1. segmenting a static gesture image into a binary hand silhouette;
2. cropping the image area to accommodate only the detected hand;
3. decomposing the image into palm and fingers.

For every hand touch, several snapshots are captured and attached to the touch point (for pseudo-code, see section 9.6). Each snapshot is converted to a binary image using a brightness threshold of 0.05, a measure that was determined based on experimenting with Microsoft Surface 1, for other devices or for environment with changes in lighting, a calibration process in the code will be necessary. Next, the snapshot that contains the biggest amount of white (meaning the biggest contact area) is selected and the other snapshots removed.

Retrieving the image of a frame during a touch event will return the whole area of the device. To guarantee that the snapshot shows only content of the touch point it belongs to, the contact area of the *touch point,* which is the bounding area of the contact, is used to set the ROI (*Region of Interest*) of the snapshot. The ROI will identify a part on the image that will be used for calculations. The ROI is defined to show only the contact area of the touch point avoiding associating of snapshots with incorrect touch points. This also allows the detection of several hands at the same time. The snapshot at first contains all the hands currently placed on the device, but by using the ROI of the touch point to crop the image, the final image will contain only the image of the hand of that touch point. As the hand is an uneven surface, intermediate touch points are detected when different parts of the hand touch on the surface of the tabletop. With the cropped snapshot containing only the binary image of the hand, the palm of the hand and the position of the fingers are detected by computing a convex hull for the white portion of the binary snapshot and its convexity defects [87]. Figure 34 illustrates how convexity defects can be used to determine fingers. Section 9.7 shows the pseudo-code of the implementation of this approach, based on [88].

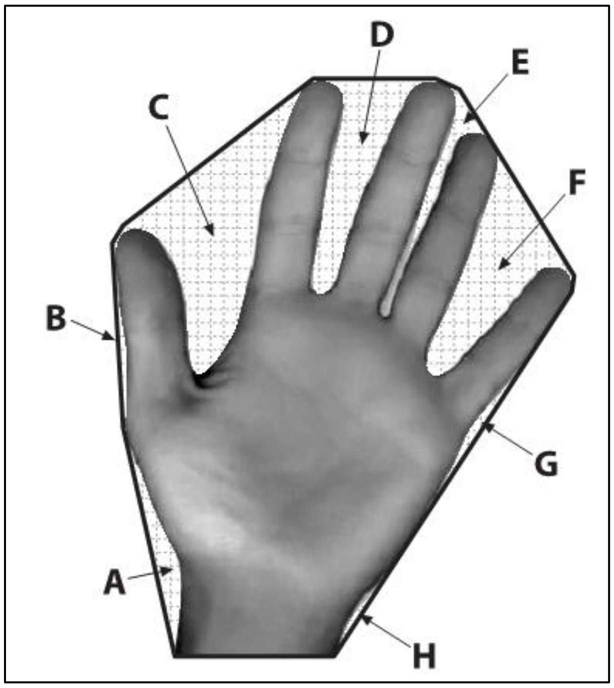


Figure 34 Convexity defects: the dark contour line is a convex hull around the hand; the gridded regions (A–H) are convexity defects in the hand contour relative to the convex hull. Extracted from [88]

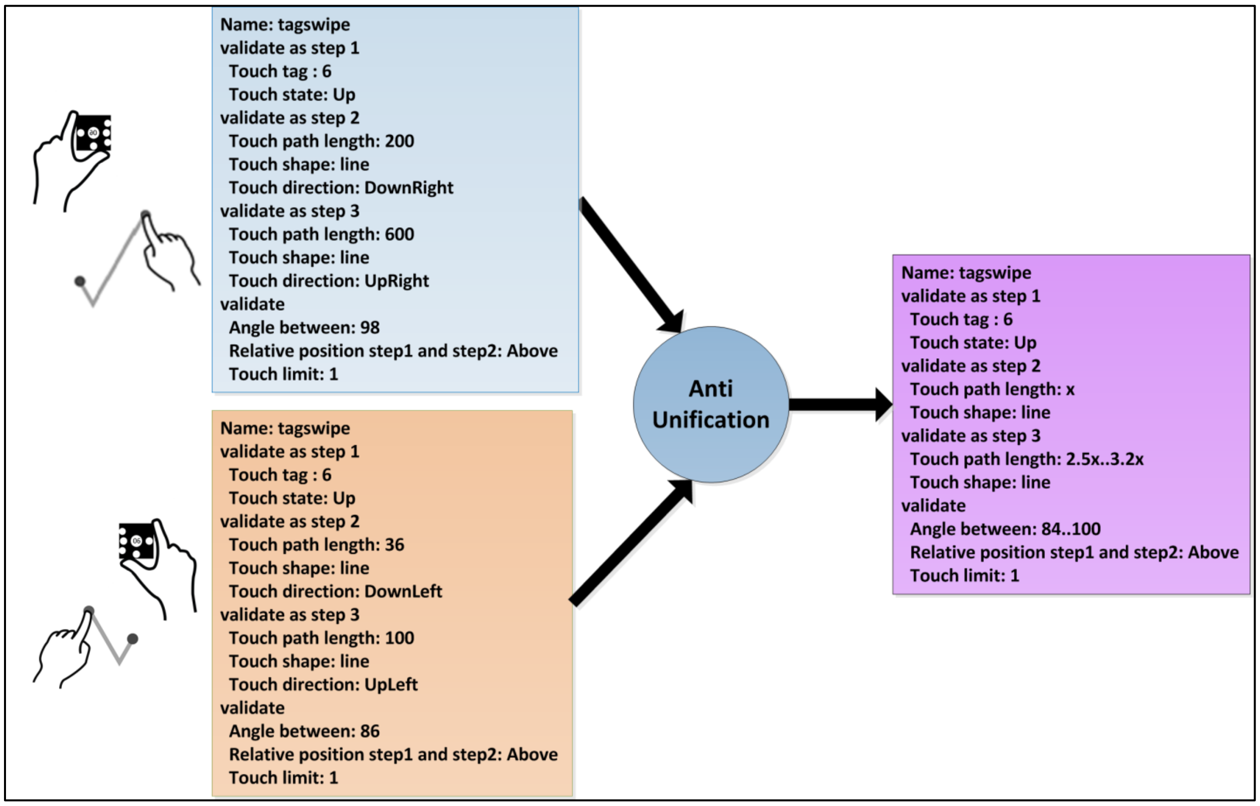
The *touch analyzer* analyzes the points, removing this redundancy in two steps: first by selecting points that are concentric to the detected *BLOB* and second by removing finger points detected in the surrounds of the detected *BLOB* with a fixed margin of 300 mm. This margin was defined based in custom heuristics using different hand sizes and establishing a value that would allow the detection of the fingers and reduce noise detection of different touch points.

5.9 Anti-unification process

To define a gesture based on samples requires a definition that is the most specific pattern among certain variations. In order to achieve this solution, IGT treats the problem as an anti-unification problem which means that it finds the most specific template or pattern between two terms, namely gesture definition of samples of gestures written in GDL. This section first provides an example to explain the overall functionality of the anti-unification process and then, explain how the algorithm works.

The example in Figure 35 shows how a designer can create a gesture that involves tangibles and a check gesture that will be recognized in both directions (avoiding the problem of defining gestures for left or right handed users, for example). The first sample in Figure 35 on the top left side, defines a gesture that is composed of a tag placed and removed from the device and a check gesture made from left to right that contains one line of 200 mm and another one of 600 mm, forming an angle of 98 degrees between them. The second sample on the top right side, shows a similar gesture but in a reduced scale and in an opposite direction (from right to left). The gesture definition generated has made three generalizations: the length of the lines is now a proportion where the second line can vary from 2.7 times to 3.2 times the length of the first line. As the direction varied, it is not contained in the definition and finally the angle can now vary from 84 to 100 degrees.

To deal with the arithmetic around the values of the primitives, formally we would have to perform anti-unification modulo the arithmetic theory. In general, anti-unification modulo theories is undecidable [79] and unfortunately also the arithmetic theory is undecidable. Therefore, we decided in our implementation to approximate this theory using the user-given sequence of the gesture parts to limit the search space for the anti-unification (and concentrating on a vaguely defined subset of the theory: multiplication and how we want to use it in a gesture definition) [79].



```
Name: tagswipe
validate as step 1
  Touch tag : 6
  Touch state: Up
validate as step 2
  Touch path length: 200
  Touch shape: line
  Touch direction: DownRight
validate as step 3
  Touch path length: 600
  Touch shape: line
  Touch direction: UpRight
validate
  Angle between: 98
  Relative position step1 and step2: Above
  Touch limit: 1
```

```
Name: tagswipe
validate as step 1
  Touch tag : 6
  Touch state: Up
validate as step 2
  Touch path length: 36
  Touch shape: line
  Touch direction: DownLeft
validate as step 3
  Touch path length: 100
  Touch shape: line
  Touch direction: UpLeft
validate
  Angle between: 86
  Relative position step1 and step2: Above
  Touch limit: 1
```

```
Name: tagswipe
validate as step 1
  Touch tag : 6
  Touch state: Up
validate as step 2
  Touch path length: x
  Touch shape: line
validate as step 3
  Touch path length: 2.5x..3.2x
  Touch shape: line
validate
  Angle between: 84..100
  Relative position step1 and step2: Above
  Touch limit: 1
```

Figure 35 Anti-unification of two samples of a gesture

In order to teach IGT a new gesture, a designer provides samples of a gesture he wants to create a definition for. For example, in Figure 35, a designer wants to create a gesture that is recognized with the placement of a specific tag and a “*check*” gesture. By providing the first sample (on top left corner of Figure 35), the *touch analyzer* will filter the *noise* touch points for the tag recognition (as explained in section 5.8.2) and perform *stroke analysis* on the *“check”* gesture (as explained in section 5.8.1). After the analysis phase, the touch points will be converted to GDL as explained in section 5.3. After being filtered by the *touch analyzer* the sample is defined in GDL. The GDL of the samples are then provided to the anti-unification algorithm (a pseudo-code of the algorithm can be seen in section 9.5).

The anti-unification process in IGT works in two main stages; first, as proposed by Nilsson in 2005 [80], it analyzes the primitives for one sample of the gesture and creates relational primitives based on relations between the different steps of that gesture. The anti-unification algorithm will take each primitive of a sample and if the primitive is numeric, it will check if that primitive exists in other steps of the same sample. If so, it will create a new primitive containing the proportion between these two steps. In the second step, the algorithm will compare the same primitives between different samples, and it will create a new gesture definition in GDL, containing primitives from the samples in which:

- literal values are constant among the samples;
- numeric values are within the threshold;
- proportional primitives are consistent among the samples;

The threshold used to verify the numeric values consistency is based on the matching accuracy parameter provided in the options box (Figure 20). The matching accuracy will affect the range of the gesture recognition. The value is calculated according to the formula in Figure 36.

Final Matching accuracy = 1 – [(Matching accuracy from option box) / 100]

Figure 36 Matching accuracy based on the value from option box

For example, for a primitive *Touch Path Length* with 200 px, values accepted in a threshold having the matching accuracy of 80% will imply that values can only vary on the spectrum from 160 px to 240 px. Reducing the matching accuracy to 30% for the same *Touch Path Length* value would imply in a gesture definition that accepts *Touch Path Length* varying from 60 px to 340 px.

Given a number n of samples of a gesture $\{x_1, x_2, \dots, x_{n-1}, x_n\}$, the algorithm creates an anti-unifier of the first two samples and then anti-unifies the created anti-unifier with the next sample to create the next anti-unified pattern. It repeats this process until it compares with the sample x_n. Based on the final pattern, the algorithm generates a gesture definition that is the most specific definition that matches all the samples provided. If the designer is not satisfied with the gesture recognition he can remove previous samples or provide new ones to change the gesture definition. When satisfied with the gesture recognition, the designer can save it and use it in ProtoActive through the process defined in sections 4.3.1.2, 4.3.1.3 and 4.4. ProtoActive assumes that it is in the best interest of the designer to provide samples that are representative of the gesture that should be used in the application. Providing samples of a gesture too similar to each other will make the resultant gesture definition too specific. However, providing gesture definitions that are too different will result in a gesture definition that is too general, thus generating false positives by recognizing undesirable interactions. The gesture definition created in IGT can be saved and it automatically becomes available in ProtoActive, where a designer can assign it to an area in a prototype page, as described in section 4.3.1.3.

CHAPTER SIX : EVALUATION

"*The choice of evaluation methodology - if any - must arise and be appropriate for the actual problem or research question under consideration*" [89]

In order to evaluate the different aspects of ProtoActive, the evaluation was conducted in three stages: first, a pilot study was conducted to evaluate the ability of ProtoActive to design ITS applications by having developers with experience in developing multi-touch applications for tabletops; after incorporating the results of the first pilot study the second pilot study was conducted with designers with experience in designing tangible applications and focused in getting qualitative feedback from the designers about using ProtoActive to design tangible applications for tabletops; the third stage was an evaluation of ProtoActive in *the wild*, providing ProtoActive to designers and have them use it for the period of at least two weeks in their projects.

6.1 Pilot study of gesture based prototypes and ProtoActive usability

A pilot user study of ProtoActive was conducted with seven participants; each one of the participants had a minimum of six months of experience developing ITS applications for academic projects. Participants were presented with a demo of ProtoActive usage that lasted on average ten minutes. The demo explained how to draw, navigate between pages in ProtoActive and how to use IGT to create a gesture. In order to reduce biasing users, no gestures were created in the system before the evaluation. A scenario proposed that participants are designers in a company that needs to create a prototype for an ITS medical application to select MRI scans. The scenario given to participants covered three main functionalities in a similar fashion as shown in Figure 15:

- a log in screen;
- selecting a scan image;
- bringing up a menu over an image to delete it.

The participants were asked to create the prototypes using ProtoActive on a Microsoft Surface and see the results by clicking "evaluate prototype" when done. According to the *Think Aloud Protocol* [81], participants were encouraged to verbalize their impressions and comment throughout their experience with ProtoActive. By the end of the evaluation, participants were asked to complete a survey that asked for their impression of using ProtoActive. Table 3 shows the mean time to complete each task.

Table 3 Average time to complete each task per user

Login page	Select image	Bring up menu and delete image
4:46	5:20	5:15

The most amount of time was spent when the participant was not satisfied with gesture recognition. Defining a gesture in IGT had usability issues regarding the information that is shown to participants. While providing a sample gesture to IGT, the only feedback that participants used for checking whether the provided sample was analyzed properly was the canvas that contained the strokes from the gesture. One of the participants that tried to look at the GDL of the gesture, said that it did not mean a lot to him and that "*it seemed fine*". Another participant mentioned that "(GDL) *it doesn't look clear enough to read it*". The participants were asked to create any gesture they thought to be appropriate for the task. Figure 37, Figure 38 and Figure 39 show the gestures created for each task and the occurrence of the gesture for that task.

With a few exceptions (e.g.: *hand*, *circle*, *square*, *check*), the gestures created were simple and usually found out of the box with frameworks such as WPF [20] (e.g.: *tap*, *swipe*).

The second biggest factor for increasing the time was due to usability issues that generated misunderstandings about the application state. E.g.: not assigning a gesture to a *gesture area*, this created the impression among participants that the application did not recognize the gesture.

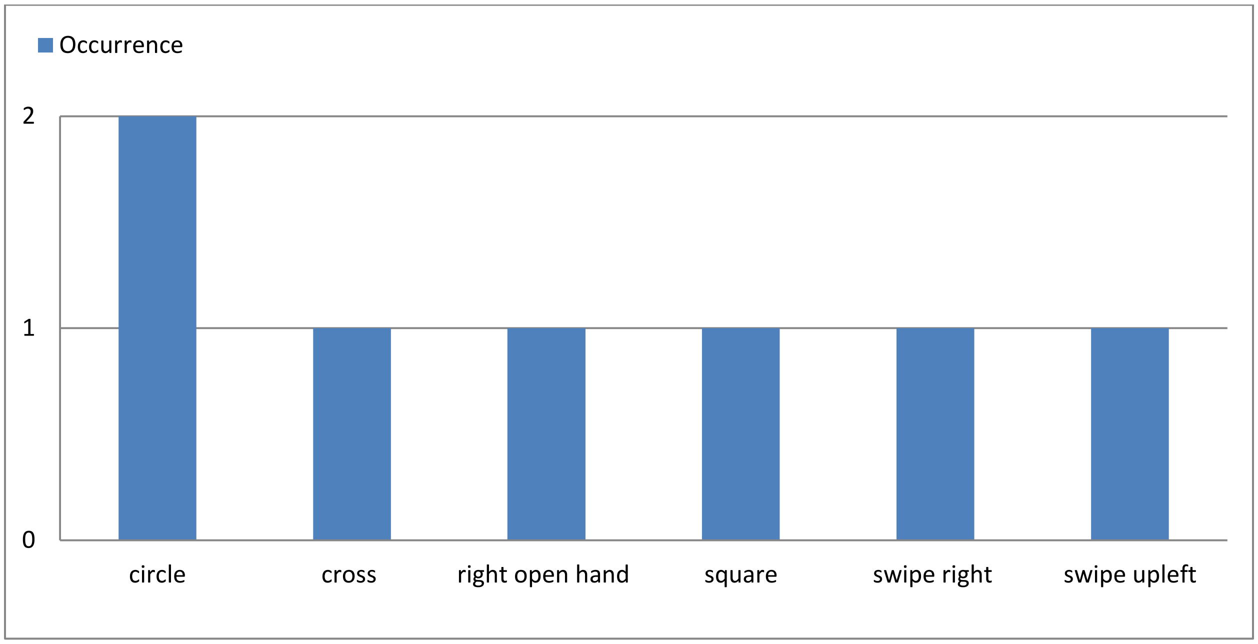


Figure 37 Gestures used for login

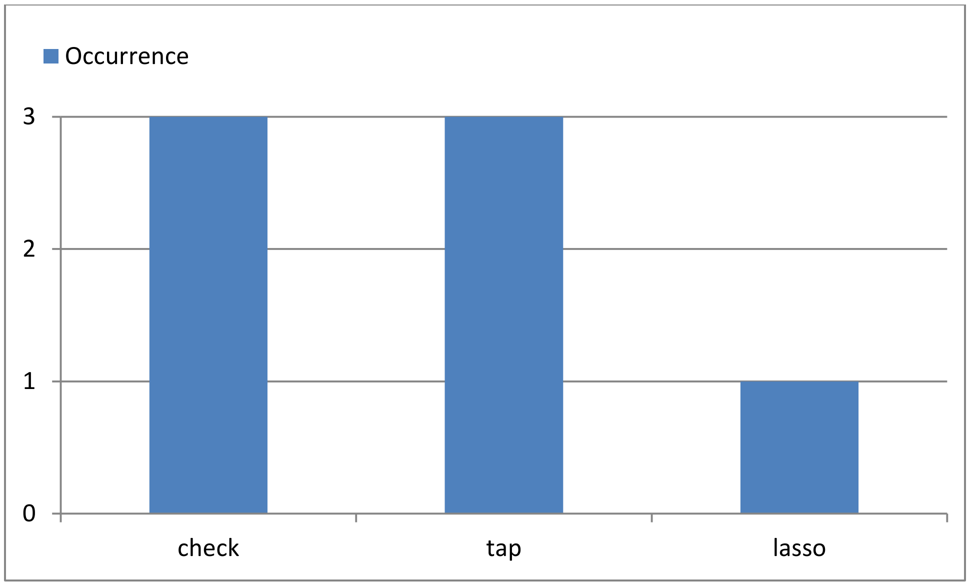


Figure 38 Gestures used for selecting an image

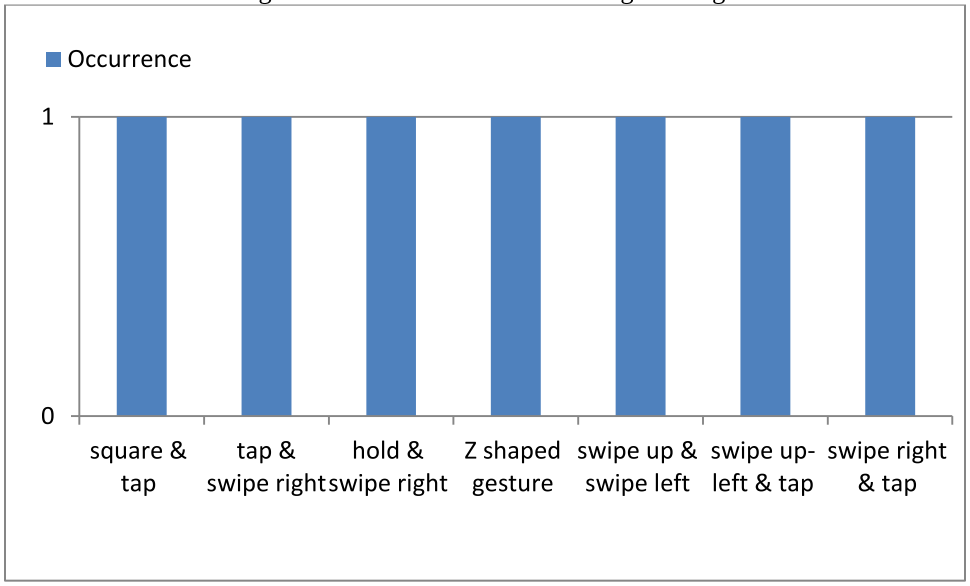


Figure 39 Gestures used to open menu over an image and delete

A task that consists of two gestures, opening a menu and deleting an image, produced a wide range of gestures as can be seen in Figure 39. This shows the potential of different interaction approaches that can be used for the same task and emphasizes the need of a tool like ProtoActive that allows designers to explore different interaction approaches but evaluating these interactions in user studies using prototypes. The survey showed that overall the participants were satisfied with both IGT and ProtoActive, with IGT having a few remarks when it appeared that a gesture was not recognized: feedback about the samples recognized for a gesture definition and problems with sketching in ProtoActive.

6.1.1 Providing feedback about samples recognized for gesture definition

The problem of showing designers how a sample was recognized is not trivial as:

- having detailed information about the recognized sample might require expertise from designers to understand it;
- not having any information won't allow designers to identify potential recognition problems.

The approach chosen by the author was to show a thumbnail of the print of the sample on the canvas (the stroke generated while providing the sample) next to the definition of the sample in GDL. A designer isn't required to read the GDL of each sample but if assumed necessary, he can look for this more detailed information of the steps recognized in the provide sample that might affect the gesture recognition.

6.1.2 Problems with sketching in ProtoActive

A problem noticed during this evaluation was caused by a design decision about the sketching features of ProtoActive. To take advantages of a multi-touch device (for the study, Microsoft Surface 1), selecting strokes, erasing and defining gesture area would need a combination of two hands to happen: while one finger stays pressing the correspondent button, the other would perform the action on ProtoActive drawing canvas (e.g.: one hand holds the selection button while the other performs a lasso on canvas to select strokes). This feature was not well accepted by participants, due to:

- hardware limitation, in some occasions an event would be miss-triggered, detecting that a finger was moved up from the device, changing the drawing mode;
- depending on the distance between the button and the place on the canvas that the action was performed, it felt uncomfortable for participants;
- in some occasions, participants would move up the finger holding the button by mistake.

The solution found by the author was to change this functionality to a regular button on the screen that doesn't need to be held.

6.2 Prototyping TUI applications

This qualitative study investigated the value of prototyping in the design of tangible applications. The study was conducted with five designers that had being involved with designing tangible applications in the past, four designers from academia and one from industry. In order to validate the proposed cycle (Figure 40) in the tangible application context, designers were asked to create a prototype in ProtoActive which was followed by giving participants some clay, printed tags and plastic toys with a tag attached to it. The aim of this study was to have participants think aloud about prototyping for tangible applications, to collect information about the value of prototyping for tangible applications and how a tool could better improve this process.

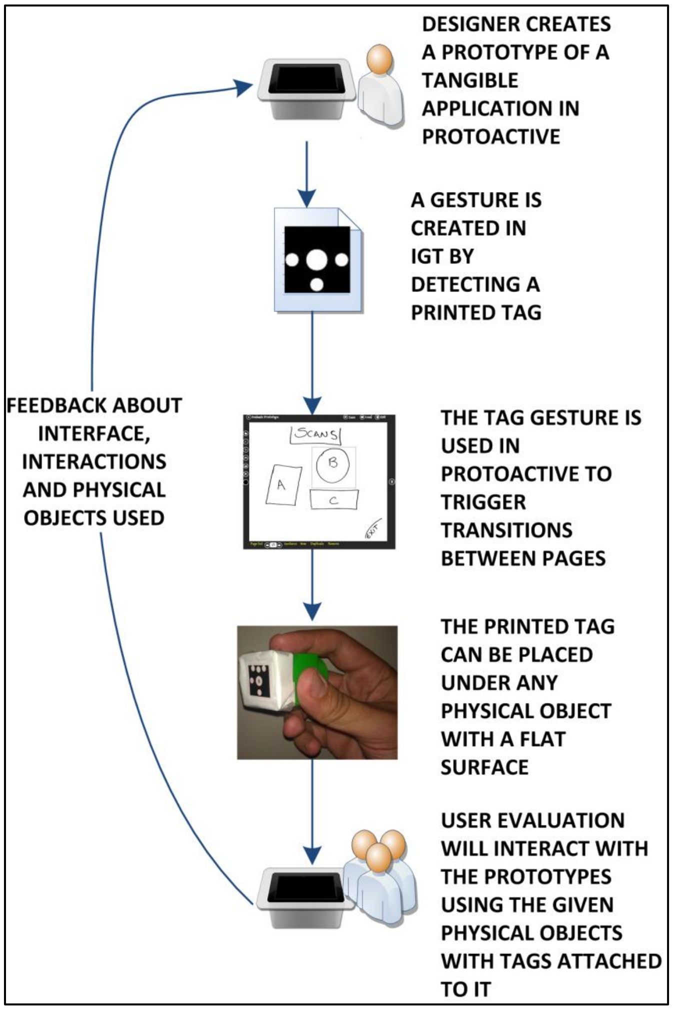

DESIGNER CREATES A PROTOTYPE OF A TANGIBLE APPLICATION IN PROTOACTIVE
A GESTURE IS CREATED IN IGT BY DETECTING A PRINTED TAG
THE TAG GESTURE IS USED IN PROTOACTIVE TO TRIGGER TRANSITIONS BETWEEN PAGES
THE PRINTED TAG CAN BE PLACED UNDER ANY PHYSICAL OBJECT WITH A FLAT SURFACE
USER EVALUATION WILL INTERACT WITH THE PROTOTYPES USING THE GIVEN PHYSICAL OBJECTS WITH TAGS ATTACHED TO IT
FEEDBACK ABOUT INTERFACE, INTERACTIONS AND PHYSICAL OBJECTS USED

Figure 40 Evaluation of a tangible application and its tangibles. Adapted from Figure 1

Participants were given a scenario where they need to design some functionalities of a *Geographical Information System* (GIS) application. Participants were asked to sketch prototypes of:

- a login screen;
- placing a specific tag that activates a weather layer on the map;
- placing another tag that activates a vegetation layer on the map.

According to the *Think Aloud Protocol* [81], participants were encouraged to verbalize their impressions and comment while creating the prototypes in ProtoActive. After creating the prototypes, a semi-structured interview was conducted. The interview covered the following points about ProtoActive:

- the importance of prototyping for tangible applications, regarding the application and the physical tangibles;
- which kind of interaction they have implemented in the tangible applications with which they were involved;
- their impressions about ProtoActive and how they think ProtoActive could have aided them in their projects.

None of the participants had previously used prototypes for their tangible applications. In all the cases no formal prototyping stage was conducted in the development of the applications. Design ideas were communicated through paper sketches and tangibles were used with a trial and error approach. The participants were impressed with the amount of design ideas that could be covered in a prototype that cost less than thirty minutes to be created with ProtoActive. When asked about how useful ProtoActive would be to quickly evaluate design ideas for tangible applications, one of the participants commented: *"coding the interface and the interactions would take forever (...) but if I would use sketches on a paper, I am not sure that I could represent it* (tangibles interactions) *just as nicely"*.

Using ProtoActive, as mentioned by the participants, consumed less time that some bad design decisions had cost in previous projects and can even help to communicate design ideas between teammates: "(to communicate ideas between teammates) *it is so much easier if you can see what you're talking about"*.

Participants found that using tangibles to interact with the applications was an important asset which becomes crucial if the final application is used on larger displays or with several users. An example given by one of the participants indicates that an approach using tangible instead of buttons would make more sense in a scenario where the interaction through buttons on the application is problematic due to positioning the buttons in a place that is not reachable by all users. This kind of problem could be detected and the solution shown to users by using ProtoActive. Figure 41 shows tangibles that were shown to participants and that could be used in the evaluation; Figure 41 compares a tree created by a participant with a fiduciary tag that was used to activate the vegetation layer on the study and a plastic toy that was also used.

Having the participants commenting about these two options allowed the researcher to understand how crucial to this stage of design the shape of the tangible is. This was mentioned by one of the participants: *"sometimes the concept is still too abstract that the shape of the tangible doesn't matter (...) but there are other cases when it might be important to differentiate, some shapes automatically represent what you want to show, for example, this is a tree and represents vegetation".*

The interviews showed that participants found that creating clay prototypes of the tangibles is a valuable asset, especially for tangibles that imply movement and require ergonomics studies. For situations where a tangible does not need any special shape, the clay did not seem necessary, and the participants chose to use tags simply attached to colored plastic toys. Participants also commented that a valuable asset of this approach is to also bring clay and printed tags for the evaluation of the prototypes with users, allowing them to make suggestions and even have them create their own clay prototypes during the prototype evaluation. As mentioned by one of the participants: "*you need to prototype it* (the tangible) *as well as it might affect the interaction*".

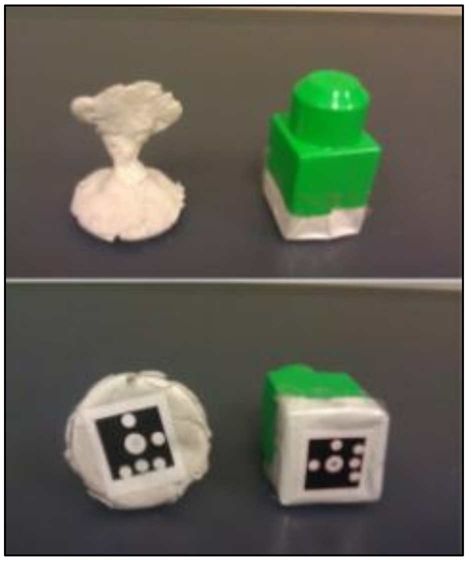


Figure 41 Clay custom tangibles and the plastic toy tangible

Regarding the mixture of hand gestures and tangibles, only one participant chose to use it, for the login screen: the participant placed a tag and made a right swipe with his finger. When asked about this mixture, participants said that users could find this mixture confusing and should rather have a clear distinction of functionalities for tangible interactions and gestures. Participants were also asked about the tangible interactions they have implemented in their previous projects. From the tangible applications previously created by the participants, the tangible interactions could be validated using ProtoActive as they were based on the detection of fiduciary markers on a tabletop, with the exception of two cases where the tangibles used were electronic devices that should also respond to the interactions and one case where a 3D movement capture was used instead of working with a multi-touch display. With the electronic device applications ProtoActive could still be used but cannot cover the interaction with the application embedded in the tangible.

Interviews were conducted with participants to better understand the needs of tangible application designers in early stages of the design process and show that low-fidelity prototyping proves to be a valuable aid in the development of tangible applications and the proposed cycle will help as a productive and useful way to evaluate interactions and tangibles for such applications.

6.3 Evaluations in the wild

A final evaluation was conducted with two participants to take a look into ProtoActive efficacy for designing applications *in the wild* [84]. This evaluation was conducted by asking UX designers in industry and in the academy to use ProtoActive in their design process. The aim of this evaluation was to have designers using ProtoActive in their own environment and to help the design of applications they care about. The evaluation was structured in two phases. First, we provided the tool installation and a brief explanation of the tool, explaining the features of the tool, in the case where this could not be done personally, a video of ProtoActive's functionality was sent to the participant. Later, when the participant spent at least two weeks with ProtoActive, the author contacted the participants individually, sending a survey (survey structure in section 9.11) and using the responses on the survey as a guide to a semi-structured interview aiming to collect data about the gestures created using ProtoActive, the application being designed by the participant and the sketches created.

6.3.1 Using ProtoActive to design a tabletop game

The first participant is a PhD candidate with no previous experience in designing ITS applications or programmatically creating gestures, but with experience in using pen and paper to prototype interfaces. The device used to create prototypes was an *Asus Eee Slate* tablet [85] with 4 GB RAM using Windows 7 and a touch-monitor supporting up to two touch points. The final application will be used in a tabletop device, but a tablet was used to for prototyping due to:

- availability of the device, as the tabletop that could be used to prototype is shared among other teammates for different projects;
- portability, as sometimes the design had to be shown or evaluated in different locations, having a tabletop would impair the evaluation process.

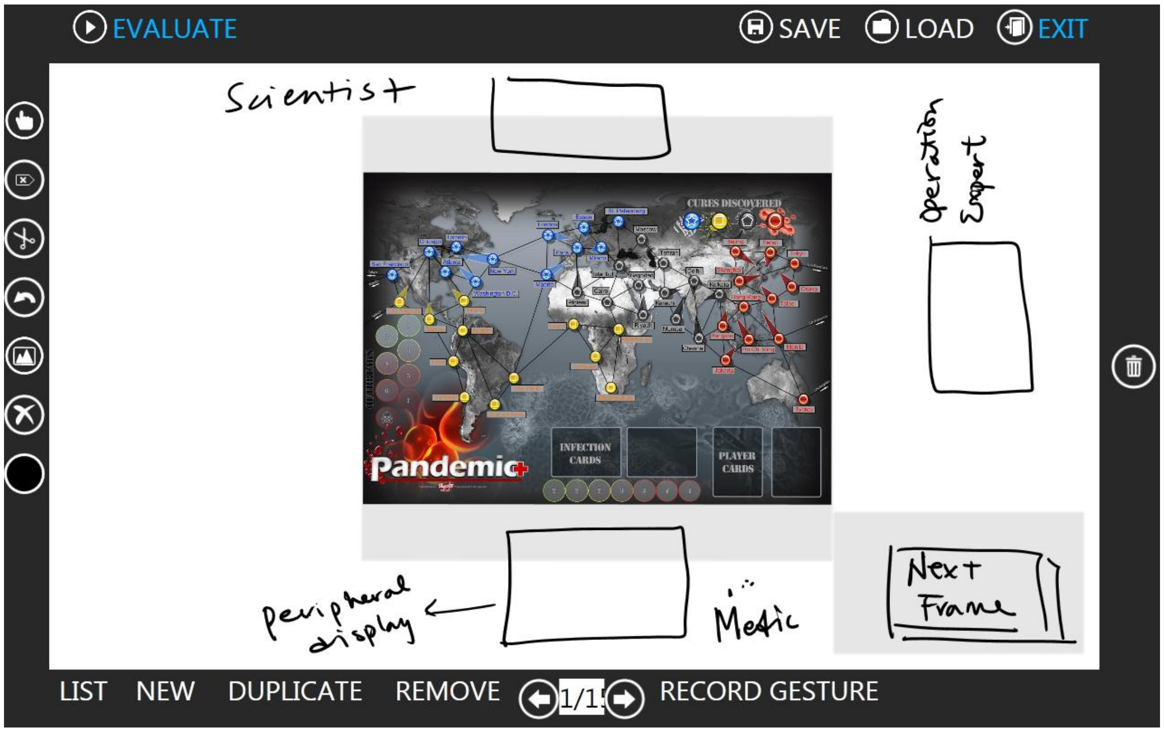


Figure 42 Prototype of a tabletop game in ProtoActive

ProtoActive was used to prototype a new version of the high automation interface for a tabletop version of the pandemic game described by Wallace *et al.* [86]. According to the participant, ProtoActive was used in the following scenario:

- using ProtoActive as a tool to brainstorm and sketch different ideas, later if needed, interactivity can be added to the sketches on ProtoActive and they can be evaluated;
- creating a prototype of small tasks and guide users to play with the prototype to elicit discussion about the interface and the interactions in it;

- prototypes were used to transmit ideas about design and interaction options, in this scenario the designer was the one interacting with the prototypes and was mostly used during meetings to communicate the design ideas to supervisors and teammates.

In total, the participant estimated to have used ProtoActive for fifteen hours spread along three weeks, generating twelve different prototypes, and having four users that evaluated the prototypes throughout three weeks. Regarding the gesture definition feature, the participant commented: *"I think the defining custom gesture functionality was pretty good. It is unclear what order to carry out actions for first time users. However, once learnt, I think it is pretty good"*. For interacting with the prototypes, the participant used pre-built gestures with the addition of two custom gestures: *two fingers hold* and *two fingers swipe*.

Figure 42 shows the main screen of one of the prototypes created for the pandemic game, illustrating how by adding images, a designer can mix different levels of refinement for a prototype. Figure 43 illustrates how the participant used ProtoActive to better illustrate specific points of a prototype. The top of Figure 43 shows how a menu will appear contextualized within the game screen; the bottom of Figure 43 shows the menu in more detail, showing how a designer can have feedback about different depth of functionalities. Also, as can be seen in Figure 43, a prototyping tool based in pre-built UI widgets would change the level of abstraction as most of the interface items in the prototypes are undefined shapes.

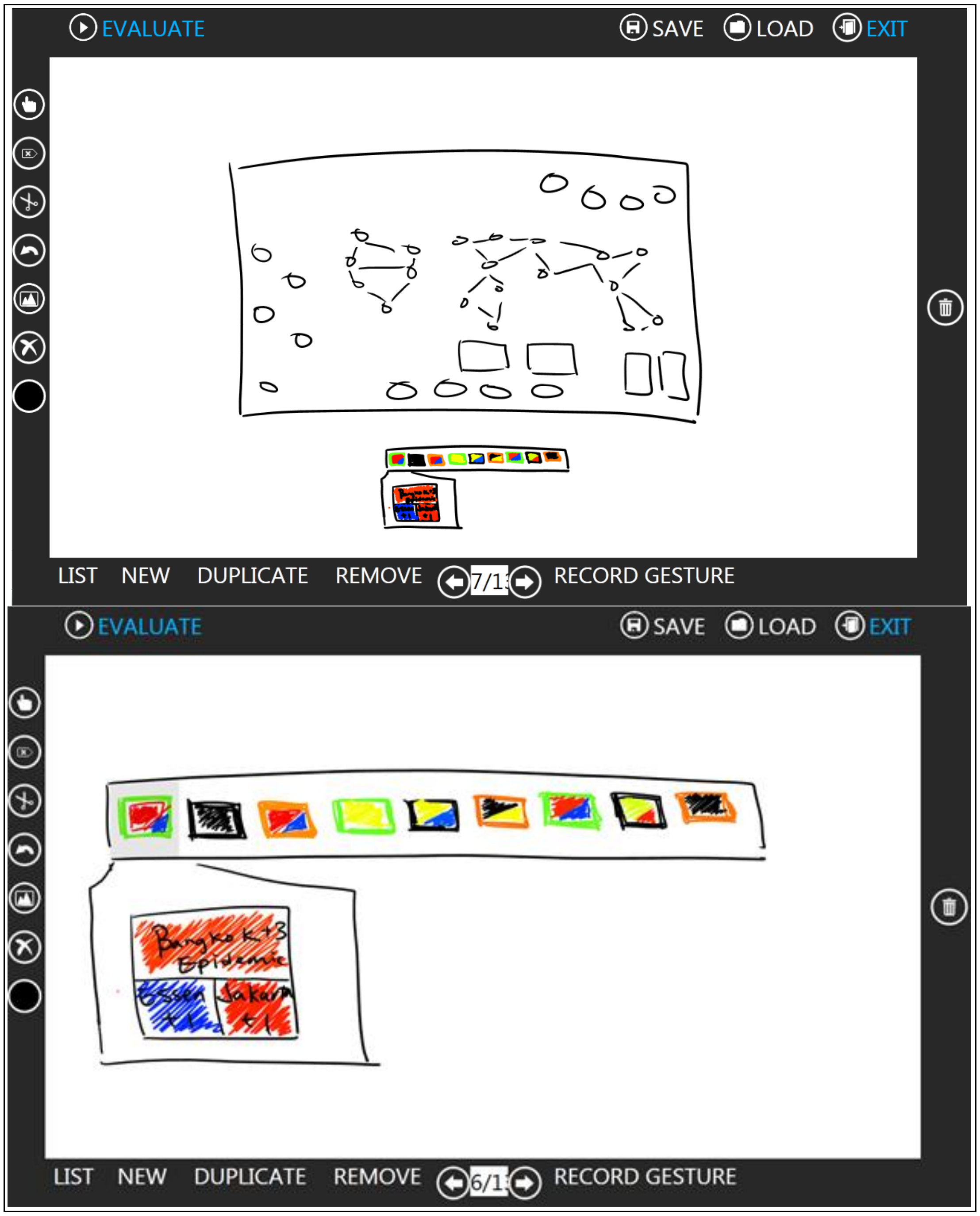


Figure 43 Using ProtoActive to provide more detail about items

When prototypes were used to transmit ideas, in some cases ProtoActive's feature of copy page was used to mock an animation by having sequential pages with small changes on the sketches.

Figure 44 shows a prototype in ProtoActive that was used to transmit ideas rather than having users interacting with it.

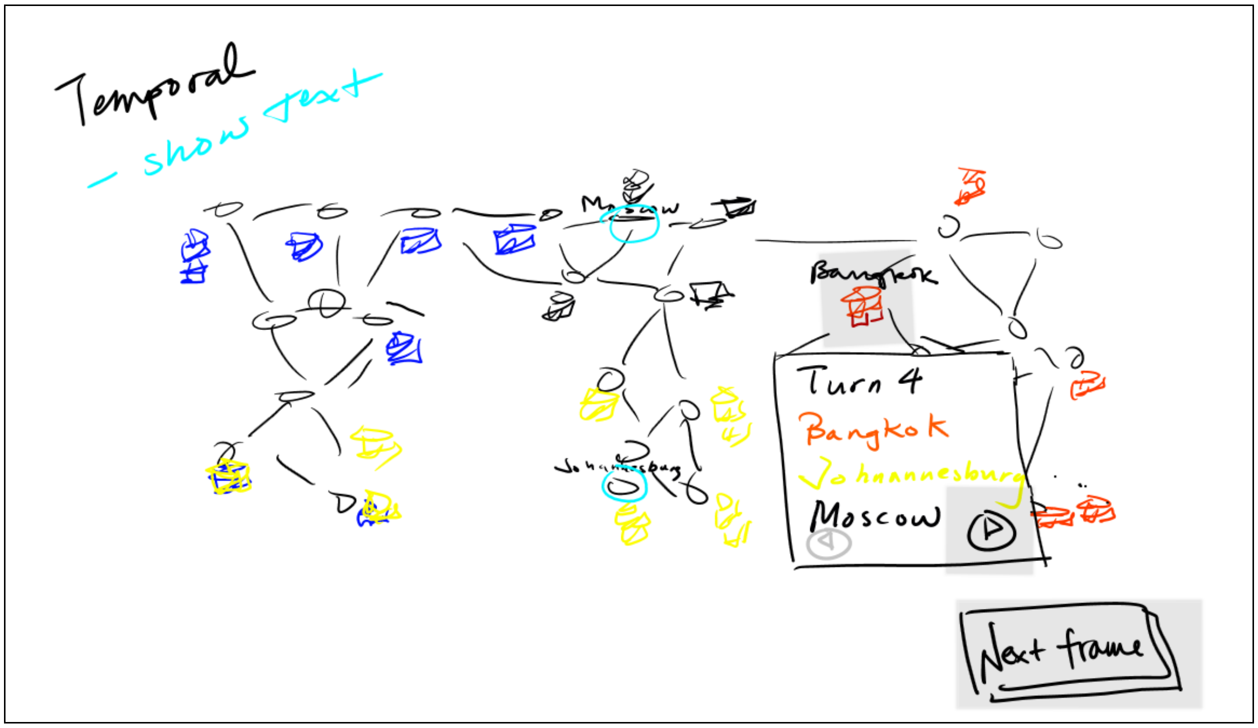


Figure 44 ProtoActive being used to transmit ideas

The “*next frame”* area set in the prototype contains a *gesture area* that responds to a *Tap* gesture that the participant used to move forward on the pages. The participant used frame concept to mock up an animation and show other people involved in the project how an animation should be shown to users.

Figure 45 shows the “*animated”* feature in detail. The “*next frame”* is used to transition between two pages of the prototype in a loop. The only difference between the pages is shown on the left and right side of Figure 45, simulating a movement on the blue concentric circles.

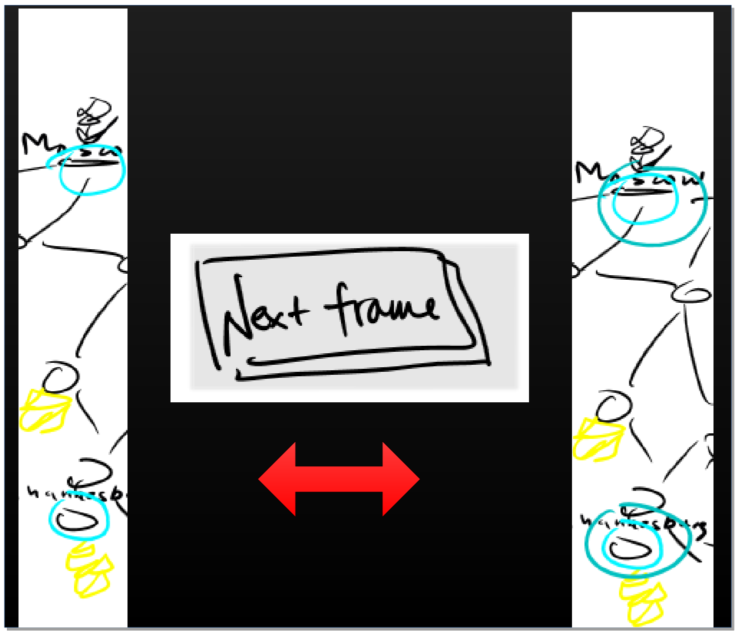


Figure 45 Detail of a prototype page, mocking up an animation

6.3.2 Using ProtoActive to design for a multi-touch device to be used in oil platforms

ProtoActive was used by a UX designer from a company to evaluate design ideas of a gesture-based application to be used in a company proprietary dual-capacitive touch display that supports two simultaneous touch points and runs on Windows 7. The display was created to resist extreme temperature conditions.

The participant had the tool for the period of three weeks and estimated to have used the tool for 4 hours. The participant is a UX designer who has eight years of experience in UX design and no experience in programmatically creating gestures for touch-devices. The participant uses low-fidelity prototypes regularly in his job and has experience with *Balsamiq Mockups* , pen/paper, *AxureRP*, *Microsoft PowerPoint* and when asked about the importance of prototyping as a tool to help the design of ITS applications, he defines it as a critical step. The designed application is a main system navigation to be used in oil platforms that will likely be used by users wearing protective gloves. Besides evaluating interface and gestures, ProtoActive will be used to study how designers in an environment with extreme temperature conditions will interact with a touch-based device: using gloves or stylus pens. Also, as working with a proprietary custom device, during the design of the applications in ProtoActive, UX designers were able to test the device capabilities and identify a problem when working with two simultaneous touch points. The overall comment from the participant was: "*Overall it's a very promising tool. We had no other tools at all for looking at gestures, so it fills a necessary void. We are unfortunately in an early development stage of our device and with ProtoActive discovered some issues with our touch screen drivers with dual touch and gestures*". For the device and software evaluation, the device will be sent to the environment where it is supposed to be used with ProtoActive prototypes to be evaluated.

```
validate as step 1
        Touch state: TouchUp
        Touch shape: Circle
        Touch direction: Right
        Touch path length: x
validate as step 2
        Touch state: TouchUp
        Touch shape: Circle
        Touch direction: Right
        Touch path length: 1.5x ..2x
validate
        Touch limit: 2
        Relative position between 1 and 2: Left
```

Figure 46 Double circle gesture defined by participant

Regarding the drawbacks and problems found using ProtoActive, the participant found that is not clear how many samples would be enough for a good gesture definition and suggested that for the anti-unified gesture definition, an image was shown illustrating a heat map of an overlap between all the gestures, where overlapping strokes would have a visualization of higher temperatures. Regarding reading GDL the participant said that he got more comfortable and could understand better the language after the time he spent using it. Figure 46 shows the definition of a "*double circle"* gesture, created by the participant to bring up a menu on the screen. The gesture consists of two fingers simultaneously making a circle shape on the screen, to the right direction. The participant's only suggestion to how the gesture definition process could be improved would be to provide a list with all the possible primitives that can be identified in GDL, but in the overall, the participant was satisfied with reading GDL and said that it was a good way to determine if a sample was properly recognized. Regarding usability issues with ProtoActive, the participant's only suggestion was to have a way to fix the position of some *gesture areas*, avoiding unintended drags on the page.

CHAPTER SEVEN : CONCLUSION

7.1 Goals achieved

This research specified three goals to be accomplished. They will be reviewed and the results found in our evaluation will be used to demonstrate how the goals were achieved.

7.1.1 How can this solution serve as a basis to help basic HCI principles such as user centered design to be followed in designing new interaction ideas?

Prototyping is one of the steps of user centered design that has been proven to be an effective way to include users early in the design process, producing products that better fit user's need by getting early feedback. ProtoActive allows designers to evaluate two aspects of ITS applications: layout ideas through sketches of the prototype and interactions through pre-built or custom gestures. The first pilot study (section 6.1) gathered different gestures that participants created to perform similar tasks. The variety of gestures created for the same task suggests that designers could benefit from such a tool as ProtoActive to evaluate different and innovative interactions. During the evaluation in the wild studies (section 6.3), ProtoActive was also used to communicate design and interaction ideas among team members, serving also as a tool to help brainstorming among team members.

ProtoActive actually allowed study participants to create and evaluate interactive prototypes of gesture-based ITS applications.

7.1.2 How does a prototyping tool that can evaluate custom gestures affect the design of touch and tag-based applications?

The first (section 6.1) and second (section 6.2) pilot studies had participants use ProtoActive and discuss its gesture creation and evaluation feature. The first study gave the same task for participants and had them create a gesture to accomplish the task in the prototype. The variety of gestures and the feedback from the participants suggest that such a feature (create custom gesture) might allow designers to innovate and try new design ideas with users due to the low cost and easiness to create and evaluate different ideas. The second pilot study evaluated ProtoActive feature of evaluating prototypes that can be interacted through tangibles, using fiduciary markers. One of the participants stated that by using such a tool, hours of development could be saved by evaluating the tag-based gesture in a prototype that took thirty minutes to be created. This shows that potential problems and design critiques might be addressed before the implementation phase. Another feedback from the second pilot study also mentioned that by being so easy to use, such a tool might also be used to explain a design idea and as a communication artifact between team members. The feedback anecdotally suggests that using ProtoActive to evaluate gestures that would be time consuming to create allows designers to experiment and evaluate ideas in an early stage. The feedback coming from experienced designers highlights ProtoActive's potential to reduce development effort for ITS applications.

7.1.3 What are the benefits of allowing designers to create custom gestures?

Allowing designers to create custom gestures allows the evaluation of different interaction ideas contained in the costs and time constraints of low-fidelity prototyping. This was shown by the evaluations (sections 6.1 and 6.2) that contained gestures that are not contained in any of the prototyping tools investigated in Chapter Three. Providing designers with ways to evaluate these gestures in the final application context (through using the custom created gestures in interactive prototypes) allows these innovative interactions to be developed following a user-centered approach as recommended by Norman and Nielsen [2].

7.2 Contributions

This thesis offers a pragmatic prototyping approach for ITS application development that is supported by two integrated tools. The first is ProtoActive, a sketch based prototyping tool for ITS applications. The main contribution of ProtoActive is to allow designers to evaluate not only the output of sketch based prototypes (namely what happens when a user wants to accomplish a task) but also the input on the prototypes and how a user wants to interact and accomplish a task. For ITS applications, interactions often use gestures and in order to allow the evaluation of these interactions, the prototypes in ProtoActive can be interacted via a pre-built set of gestures or through customized gestures created with an embed gesture learner tool, which is the thesis second contribution: IGT.

IGT uses samples of a gesture performed on the device to create a gesture definition that can recognize all the samples provided for a specific gesture. The novelty of IGT relies on the unique anti-unification approach used to identify all the common aspects between the samples of the gestures thus creating a gesture definition that is the most specific template between the samples.

The gestures created in IGT are available in external files that can be used by any other application using *Gesture Toolkit* framework to recognize gesture.

7.3 Limitations of the thesis

As for the main contributions, the limitations of the solution proposed in this thesis can be explained in terms of its two main features: prototyping and learning gestures.

7.3.1 Prototyping

The different evaluations showed that ProtoActive fills the needs for creating prototypes for ITS applications, but it arguably has limited support for gathering data during evaluation. Designers often rely on their own equipment to record the video and audio of evaluation sessions of the prototypes. ProtoActive support for evaluation relies on the same as paper prototyping. Due to the easiness to create prototypes and to manipulate the tool: using a finger or a stylus pen to draw, users with the support of designers can suggest and even make modifications to the prototype during the evaluation session.

7.3.2 Learning gestures

IGT and ProtoActive use *Gesture Framework* [10] for gesture recognition and definition. This means that some of the limitations in *Gesture Toolkit* are inherited. While the GDL supports multi-step gestures, it is currently limited to gestures with sequential steps and that need to fit in the primitives described in Chapter Five. The feature to allow the gesture recognizer to break the gesture into parts facilitates the sequential process but it requires some experience from the designer to decide if dividing a gesture into steps or not will generate the best gesture definition for its needs.

7.4 Limitations of the studies

The two first studies were designed to evaluate separate aspects of the current study. A final evaluation had a more horizontal approach to see the user's feedback after using ProtoActive for their projects and for a more extend period. The author recognizes that a comprehensive user study involving more participants will provide more insights about ProtoActive.

7.5 Future Work

The usability issues commented on the evaluation process should be addressed for a next version of ProtoActive. These design critiques are:

- for a large display, the buttons might be too far from the user;

- the shape of the *gesture area* concerned some students in scenarios such as a circular button;
- depending on the background image set, a *gesture area* might be hard to notice;
- the color picker feature might suffer from *"fat finger"* problem, where the area to click is too small for a finger;
- allow *gesture areas* to contain other *gesture areas*, allowing a stacking mechanism that will group *gesture areas.* The current state of the application does not allow *gesture areas* to be placed on top of each other;

ProtoActive and IGT's method of defining and recognizing gestures could be improved to avoid two main problems detected during the evaluation:

- unreliable hardware: misrecognition of touch-points when providing samples to IGT will generate inconsistent samples that might not represent the gesture a designer is trying to teach the tool;
- conflict during gesture recognition, as a *gesture area* can support multiple gestures, a conflict mechanism should detect potential conflicts and warn designers;

The evaluation of ProtoActive might also be improved by including more extensive evaluations (including evaluation in the wild of tangible based applications and participating in the whole cycle of an application development) and as well as use in real projects as further evaluations. Using ProtoActive for designing real world applications might allow the author to gather feedback from the final users of the application which will give a better understanding of ProtoActive's impact in the final product.

The future of ProtoActive looks promising as after the evaluations in the wild it is being considered to be used for one of the companies that participated in the study to be used to evaluate prototypes with users in oil platforms for further evaluations. It is also being considered by other participants and university professors to be used in an educational context: as a prototyping tool to be used by students in an HCI for tabletops class.

REFERENCES

1. Norman, D.,A. (2007). The Design of Future Things. Ed. Basic Books.
2. Norman, D., Nielsen, J. (2010). Gestural interfaces: a step backward in usability. Interactions, vol 17, issue 5.
3. Hesselmann, T., & Boll, S. (2011). SCIVA: designing applications for surface computersEICS 2011, 191-196.
4. Moggridge, B. (2007). Designing Interactions. Chapter 10- People and Prototypes. MIT Press, Cambridge, MA.
5. Hinrichs, U., & Carpendale, S. (2011). Gestures in the Wild : Studying Multi-Touch Gesture Sequences on Interactive Tabletop Exhibits. CHI 2011, Pages 3023-3032. May 7–12, 2011.
6. Wobbrock, J. O., Morris, M. R., & Wilson, A. D. (2009). User-defined gestures for surface computing. *Proceedings of CHI ’09*. Pages 1083-1092. New York, New York, USA: ACM Press.
7. Windows Presentation Foundation, Available at http://msdn.microsoft.com/en-us/library/ms754130.aspx, Accessed December 2010.
8. Microsoft Surface User Experience Guidelines. Available at http://www.microsoft.com/en-ca/download/confirmation.aspx?id=19410. Accessed in November 2012.
9. GestureWorks, a multitouch application framework for Adobe Flash and Flex. 2010. Available at http://gestureworks.com/ accessed in February 2012.
10. Khandkar, S. H., & Maurer, F. (2010). A Domain Specific Language to Define Gestures for Multi-Touch Applications. DSM:10, 17-OCT-2010, Reno, USA.

11. Lao, S., Heng, X., Zhang, G., Ling, Y., Wang, P. (2009). A gestural interaction design model for multi-touch displays. In Proceedings of the 23rd British HCI Group Annual Conference on People and Computers: Celebrating People and Technology (BCS-HCI '09). British Computer Society, Swinton, UK, UK, 440-446.

12. Allan Christian Long, Jr., James A. Landay, and Lawrence A. Rowe. (1999). Implications for a gesture design tool. In Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI '99). ACM, New York, NY, USA, 40-47.

13. Kent Lyons, Helene Brashear, Tracy Westeyn, Jung Soo Kim, and Thad Starner. 2007. GART: the gesture and activity recognition toolkit. In Proceedings of the 12th international conference on Human-computer interaction: intelligent multimodal interaction environments (HCI'07), Julie A. Jacko (Ed.). Springer-Verlag, Berlin, Heidelberg, 718-727.

14. Kin, K., Hartmann B., DeRose T., Agrawala M.. Proton: Multitouch Gestures as Regular Expressions. CHI 2012, ACM 978-1-4503-1015-4/12/05.

15. Beryl Plimmer, Rachel Blagojevic, Samuel Hsiao-Heng Chang, Paul Schmieder, and Jacky Shunjie Zhen. 2012. RATA: codeless generation of gesture recognizers. In Proceedings of the 26th Annual BCS Interaction Specialist Group Conference on People and Computers (BCS-HCI '12). British Computer Society, Swinton, UK, UK, 137-146.

16. Wiethoff, A., Schneider, H., Rohs, M., & Butz, A. Greenberg, S. (2012). Sketch-a-TUI: low cost prototyping of tangible interactions using cardboard and conductive ink. Embodied Interaction, 1(212), 309-312.

17. Rudd, J., Stern, K., Isensee, S. (1996). Low vs. high-fidelity prototyping debate, interactions, v.3 n.1, p.76-85.

18. Sefelin, R., Tscheligi, M., Giller, V. (2003). Paper prototyping - what is it good for?: a comparison of paper and computer-based low-fidelity prototyping, CHI '03 extended abstracts on Human factors in computing systems, April 05-10, 2003, Ft. Lauderdale, Florida,USA.

19. Virzi, R.A., Sokolov, J.L., Karis, D. (1996). Usability problem identification using both low- and high-fidelity prototypes, Proceedings of the SIGCHI conference on Human factors in computing systems: common ground, p.236-243, April 13-18, 1996, Vancouver, British Columbia, Canada.

20. McCurdy, M., Connors, C., Pyrzak, G., Kanefsky, B., Vera, A. (2006). Breaking the fidelity barrier: an examination of our current characterization of prototypes and an example of a mixed-fidelity success. In Proceedings of the SIGCHI conference on Human Factors in computing systems (CHI '06), Rebecca Grinter, Thomas Rodden, Paul Aoki, Ed Cutrell, Robin Jeffries, and Gary Olson (Eds.). ACM, New York, NY, USA, 1233-1242.

21. Youn-Kyung Lim, Stolterman, E., Tenenberg, J.. (2008). The anatomy of prototypes: Prototypes as filters, prototypes as manifestations of design ideas. ACM Trans. Comput.-Hum. Interact. 15, 2, Article 7 (July 2008), 27 pages.

22. Derboven, J., Roeck, D. D., & Verstraete, M. (2010). Low-Fidelity Prototyping for Multi-Touch Surfaces. . Presented in the workshop Engineering Patterns for Multi-Touch Interfaces held in EICS 2010, 2nd edition, Berlin, Germany.

23. Krippendorff, K. (2006). The Semantic Turn: A New Foundation for Design. Taylor & Francis, Boca Raton, FL.

24. Bill Buxton. (2007). Sketching User Experiences: Getting the Design Right and the Right Design. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

25. Memmel, T., Gundelsweiler, F., Reiterer, H..(2007). Agile human-centered software engineering. In Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI...but not as we know it - Volume 1 (BCS-HCI '07), Vol. 1. British Computer Society, Swinton, UK, UK, 167-175.

26. Constantine, L. L. (2004). Beyond user-centered design and user experience: Designing for user performance. Cutter IT Journal, 17, 2.

27. Microsoft Surface 1 SDK. Available at http://msdn.microsoft.com/en-us/library/ee804767(v=surface.10).aspx . Accessed in November 2012.

28. Microsoft Surface 2 SDK. Available at http://msdn.microsoft.com/en-us/library/ff727815.aspx. Accessed in November 2012.

29. Mitra, S.; Acharya, T.; , "Gesture Recognition: A Survey," Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on , vol.37, no.3, pp.311-324, May 2007.

30. Suzanne Robertson and James Robertson. (2006). Mastering the Requirements Process (2nd Edition). Chapter 12. Addison-Wesley Professional.

31. Van den Bergh, J. et al.., GRIP: (2011). get better results from interactive prototypes. In Proc. Engineering Interactive Computing Systems 2011, ACM Press 143-148.

32. Rudd, J., Stern, K., and Isensee, S. Low vs. high fidelity prototyping debate. Interactions, 3, 1 (1996), 76-85.

33. Marvin V. Zelkowitz. (1984). A taxonomy of prototype designs. SIGSOFT Softw. Eng. Notes 9, 5 (October 1984), 11-12.

34. North, C., Dwyer, T., Lee, B., Fisher,D., Isenberg,P., Robertson,G., Inkpen, K.. (2009) Understanding Multi-touch Manipulation for Surface Computing. In Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction: Part II (INTERACT '09), Springer-Verlag, Berlin, Heidelberg, 236-249.

35. Morris, M., R., Wobbrock,J., O., Wilson., A.,D. (2010). Understanding users' preferences for surface gestures. In Proceedings of Graphics Interface 2010 (GI '10). Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 261-268.

36. Jens Gerken, Hans-Christian Jetter, Toni Schmidt, and Harald Reiterer. 2010. Can "touch" get annoying?. In ACM International Conference on Interactive Tabletops and Surfaces (ITS '10). ACM, New York, NY, USA, 257-258.

37. Holzmann, C. , Vogler, M.. (2012). Building interactive prototypes of mobile user interfaces with a digital pen. In Proceedings of the 10th asia pacific conference on Computer human interaction (APCHI '12). ACM, New York, NY, USA, 159-168.

38. Marco de Sá and Carriço, L.. (2008). Lessons from early stages design of mobile applications. In Proceedings of the 10th international conference on Human computer interaction with mobile devices and services (MobileHCI '08). ACM, New York, NY, USA, 127-136.

39. Russ Unger and Carolyn Chandler. (2012). A Project Guide to UX Design: For User Experience Designers in the Field or in the Making (2nd ed.). New Riders Publishing, Thousand Oaks, CA, USA.

40. Sinha, A.K., Landay,J.A. (2003). Capturing user tests in a multimodal, multidevice informal prototyping tool. In Proceedings of the 5th international conference on Multimodal interfaces (ICMI '03). ACM, New York, NY, USA, 117-124.

41. J. David Smith and T. C. Nicholas Graham. (2010). Raptor: sketching games with a tabletop computer. In Proceedings of the International Academic Conference on the Future of Game Design and Technology (Futureplay '10). ACM, New York, NY, USA, 191-198.

42. Željko Obrenovic and Jean-Bernard Martens. 2011. Sketching interactive systems with sketchify. ACM Trans. Comput.-Hum. Interact. 18, 1, Article 4 (May 2011), 38 pages.

43. Vinícius C. V. B. Segura, Simone D. J. Barbosa, and Fabiana Pedreira Simões. 2012. UISKEI: a sketch-based prototyping tool for defining and evaluating user interface behavior. In Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI '12), Genny Tortora, Stefano Levialdi, and Maurizio Tucci (Eds.). ACM, New York, NY, USA, 18-25.

44. Lin, J.. A visual language for a sketch-based UI prototyping tool. (1999). In CHI '99 extended abstracts on Human factors in computing systems (CHI EA '99). ACM, New York, NY, USA, 298-299.

45. Newman, M.W., Lin, J., Hong, J. I. , Landay. J.A.. (2003). DENIM: an informal web site design tool inspired by observations of practice. Hum.-Comput. Interact. 18, 3 , 259-324.

46. Bailey, B.P., Konstan, J.A., Carlis, J.V. (2001). DEMAIS: designing multimedia applications with interactive storyboards. In Proceedings of the ninth ACM international conference on Multimedia (MULTIMEDIA '01). ACM, New York, NY, USA, 241-250.

47. Hosseini-Khayat, A., Seyed, T., Burns, C., Maurer, F. (2011). Low-Fidelity Prototyping of Gesture-based Applications. EICS'11,Pages 289-294. June 13–16, 2011, Pisa, Italy.

48. Webification – 23 best user interface design/website wireframing tools. Available at http://webification.com/best-mockup-design-tools-roundup. Accessed October 2012

49. First Web Designer - 18 Wireframing, Mockup And Prototyping Tools To Plan Designs. Available at http://www.1stwebdesigner.com/design/wireframing-mockup-prototyping-tools-plan-designs/. Accessed October 2012

50. SitePoint - 16 Design Tools for Prototyping and Wireframing. Available at http://www.sitepoint.com/tools-prototyping-wireframing/. Accessed October 2012.

51. Deziner Folio.com - 14 Prototyping and Wireframing Tools for Designers. Available at http://www.dezinerfolio.com/2011/02/21/14-prototyping-and-wireframing-tools-for-designers. Accessed October 2012.

52. Balsamiq – Available at www.balsamiq.com. Accessed July 2012

53. Pencil: Add-on for Mozilla Firefox. Available at https://addons.mozilla.org/en-US/firefox/addon/pencil/. Accessed October 2012.

54. iPlotz: Wireframes, mockups and prototyping for websites. Available at http://iplotz.com/. Accessed October 2012.

55. Axure RP: Interactive wireframe software and mockup tool. Available at http://www.axure.com/. Accessed October 2012.

56. Mockingbird: Wireframes on the fly. Available at https://gomockingbird.com/. Accessed October 2012.

57. Microsoft Sketchflow. Available at http://www.microsoft.com/expression/products/sketchflow_overview.aspx. Accessed March 2012.

58. ForeUI: Easy to use UI prototyping tool. Available at http://www.foreui.com/. Accessed October 2012.

59. Proto.io : Silly-fast mobile prototyping. Available at http://proto.io/. Accessed October 2012.

60. S. Damaraju and A. Kerne. Multitouch Gesture Learning and Recognition System. In Poster session presented at TABLETOP ’08. Tabletop 2008, 2008.

61. Florian Echtler and Andreas Butz. 2012. GISpL: gestures made easy. In Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction (TEI '12), Stephen N. Spencer (Ed.). ACM, New York, NY, USA, 233-240.

62. JSON - Available at : http://www.json.org/ . Accessed in September of 2012.

63. C. Scholliers, L. Hoste, B. Signer, and W. De Meuter. Midas: a declarative multi-touch interaction framework. In Proceedings of the ?fth international conference on Tangible, embedded, and embodied interaction, TEI ’11, pages 49–56, New York, NY, USA, 2011. ACM.

64. D. Kammer, J. Wojdziak, M. Keck, R. Groh, and S. Taranko. Towards a formalization of multi-touch gestures. In ACM International Conference on Interactive Tabletops and Surfaces, ITS ’10, pages 49–58, New York, NY, USA, 2010. ACM.

65. Lisa Anthony and Jacob O. Wobbrock. 2012. $N-protractor: a fast and accurate multistroke recognizer. In Proceedings of the 2012 Graphics Interface Conference (GI '12). Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 117-120.

66. Bobick, A. F., & Wilson, A. D. (1997). A state-based approach to the representation and recognition of gesture. *Pattern Analysis and Machine Intelligence, IEEE Transactions on, 19*(12), 1325–1337.

67. Khandkar, S. H., Sohan, S. M., Sillito, J., & Maurer, F. (2010). Tool Support for Testing Complex Multi-Touch Gestures. ITS '10.

68. Alcantara, T., Denzinger, J. , Ferreira, J. , Maurer, F. (2012). Learning gestures for interacting with low-fidelity prototypes. RAISE 2012, Pg 32-36.

69. Cottrell, R., Chang, J. J. C., Walker, R. J., & Denzinger, J. (2007). Determining detailed structural correspondence for generalization tasks. *Proceedings of ESEC-FSE ’07,* Pages 165-174. New York, New York, USA.

70. Freeman, D., Benko, H., Morris, M. R., Wigdor, D., & Way, O. M. (2009). ShadowGuides : Visualizations for In-Situ Learning of Multi-Touch and Whole-Hand Gestures. ITS '09: Proceedings of the ACM ITS '09.

71. Yu, N., Chan, L., Lau, S., & Tsai, S. (2011). TUIC: enabling tangible interaction on capacitive multi-touch displays. Proceedings of the, 2995-3004.

72. Esteves, A. (2012). Designing tangible interaction for embodied facilitation. Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction - TEI ’12, 395. doi:10.1145/2148131.2148231

73. Esteves, A., & Oakley, I. (2011). Informing design by recording tangible interaction. Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems - CHI EA ’11, 2077. doi:10.1145/1979742.1979893

74. Maribeth Gandy, Brian Jones, Scott Robertson, Tiffany O`Quinn, and Amos Johnson. (2009). Rapidly Prototyping Marker Based Tangible User Interfaces. In Proceedings of the 3rd International Conference on Virtual and Mixed Reality: Held as Part of HCI International 2009 (VMR '09), Randall Shumaker (Ed.). Springer-Verlag, Berlin, Heidelberg, 159-168.

75. Klemmer, S.R., Li, J., Lin, J., & Landay, J. (2004). Papier-Mâché: Toolkit Support for Tangible Input. *proceedings ACM CHI 2004, Vienna, Austria, April 24.*

76. Calgary UX. Available at http://calgaryux.com/. Accessed on November 2012.

77. Nielsen, Jakob, and Landauer, Thomas K.: "A mathematical model of the finding of usability problems," Proceedings of ACM INTERCHI'93 Conference (Amsterdam, The Netherlands, 24-29 April 1993), pp. 206-213.

78. Chang, C., Chen, J., Tai, W., & Han, C. (2006). New approach for static gesture recognition. Journal of information science, 1057, 1047-1057

79. Jochen Burghardt. (2005). E-generalization using grammars. Artif. Intell. 165, 1 (June 2005), 1-35.

80. Nilsson, N. J. (2005). Introduction to machine learning an early draft of a proposed textbook Department of Computer Science. Stanford University,Stanford, CA 94305.

81. Lethbridge, T. C., & Sim, S. E. (2005). Studying software engineers: Data collection techniques for software field studies. Empirical Software Engineering, 10(3), 311–341. Springer.

82. 230i, SMART Technologies. Available at https://smarttech.com/Support/Browse+Support/Product+Index/Hardware+Products/SMART+Table/230i. Accessed December 2012.

83. Microsoft Powerpoint. Available at http://office.microsoft.com/en-ca/powerpoint/. Accessed December 2012.

84. Johnson,R., Rogers, Y., van der Linden, J., Bianchi-Berthouze, N. (2012). Being in the thick of in-the-wild studies: the challenges and insights of researcher participation. In Proceedings of CHI '12. ACM, New York, NY, USA, 1135-1144.

85. Asus Eee Slate EP121 - The world's most powerful tablet device. Available http://www.asus.com/Eee/Eee_Pad/Eee_Slate_EP121/ . Acessed October 2012.

86. Wallace, J.R., Pape, J., Yu-Ling Betty Chang, McClelland,P.J., Graham,T.C.N., Scott, S.D. and Hancock, M. (2012). Exploring automation in digital tabletop board game. In Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work Companion (CSCW '12). ACM, New York, NY, USA, 231-234.

87. K. Homma and E.-I. Takenaka. (1985). An image processing method for feature extraction of space-occupying lesions. Journal of Nuclear Medicine 26: 1472–1477.

88. Bradski, G., Kaehler, A. (2008). Learning OpenCV. O'Reilly. Pg 258-260.

89. Greenberg S., Buxton, B. (2008). Usability evaluation considered harmful (some of the time). In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08). ACM, New York, NY, USA, 111-120.

90. Learn Dreamweaver CS4. Available at http://tv.adobe.com/show/learn-dreamweaver-cs4/ . Accessed in November of 2012.

CHAPTER NINE : APPENDIX

9.1 Finding lines in a continuous stroke

```
public  <List of TouchPoints> FindLines(TouchPoints points)
        {
            switch (NoiseReduction)
            {
                case NoiseReductionType.Low: increment = LOW_INCREMENT;
                    break;
                case NoiseReductionType.Medium: increment = MEDIUM_INCREMENT;
                    break;
                case NoiseReductionType.High: increment = HIGH_INCREMENT;
                    break;

            }
            string sPreviousDirection = "";
                     int idxIni = 0;
            for (i = increment; i < points.Count; i = i + increment)
            {
                StylusPoint p1 = simplerPoints[i - 1];
                StylusPoint p2 = simplerPoints[i];
                double slope = GetSlopeBetweenPoints(p1, p2);
                string sDirection = SlopeToDirection(slope);

                if (sPreviousDirection == "")
                    sPreviousDirection = sDirection;

                if (sPreviousDirection != sDirection)
                {
                                  TouchPoints newLine = CreateTouchPoints()
                                  {
                                         Points = points.GetRange(idxIni, i),
                                         Action = Move
                                  }
                                  result.Add (newLine);
                    lastDirection = sCurrentDirection;
                    sPreviousDirection = sDirection;
                    idxIni = i;
                }
            }
            // If nothing was added then, there is only one step which is the whole
gesture
            if (result.Count == 0)
                result.Add(points);
            // If there are several points after the last recognized step add the last
remaining points as a new step
            else if (i > idxIni)
            {
               TouchPoints newLine = CreateTouchPoints()
                                  {
                                         Points = points.GetRange(idxIni, i),
                                         Action = Up
                                  }
                         result.Add (newLine);
            }

            return result;
        }
```

9.2 Removing noise touch points for tag recognition

```
private <List of TouchPoints>
removeConcentricBlobs(int pointIndex, <List of TouchPoints> > points)
        {
            TouchPoint p1 = points[pointIndex];
            if (!p1.isFinger)
            {
                var concentricBlobs =
                     from p in points
                     where  p.Action == p1.Action
                     && p.Tag == null
                     && p != p1
                     && p.isFinger == false
                     && (GetDistanceBetweenPoints(p1.Position, p.Position) <= REDUNDANCY)
                                   select p;
              <List of TouchPoints> temp = concentricBlobs.ToList();

                foreach (TouchPoint2 point in temp)
                {
                   if (GetSmallerBound(p1, point) == p1)
                   {
                       points.Remove(p1);
                       p1 = point;
                   }
                   else
                       points.Remove(point);
                }
            }

            pointIndex++;
            if ((points.Count > 1)&&(pointIndex < points.Count))
                return removeConcentricBlobs(pointIndex, points);
            else
                return points;
        }
```

9.3 Creating valid primitives from touch points

```
public static GDL TouchPointsToPrimitives(FilteredTouchPoints allpoints)
        {

            primitives = GetAllPrimitivesFromGestureToolkit();

            if (PointAnalyzer.BreakIntoSteps)
                allpoints = PointAnalyzer.FindLines(allpoints);

            allpoints = PointAnalyzer.analyzeTags(allpoints);
            allpoints = PointAnalyzer.removeHandRedundancy(allpoints);

            var simple = from p in primitives
                                    where p.isComplex() == false
                                    select p;

            var complex= from p in primitives
                                     where p.isComplex() == true
                                     select p;
            ValidPrimitives simplePrimitives = loadPrimitives(allpoints, simple);
            ValidPrimitives complexPrimitives = loadPrimitives(allpoints, complex));

            samplePrimitives.AddRange(simplePrimitives);
            samplePrimitives.AddRange(complexPrimitives);

            return samplePrimitives;
        }
```

9.4 Extended grammar for GDL

```
module GDL
{
     language GDL
    {

syntax PrimitiveCondition
= r:TouchState => r | r:TouchTime => r | r:TouchLimitRule => r | r:TouchPathLength => r|
r:TouchStepRule => r| r:ClosedLoop => r | r:DistanceBetweenPoints=> r | r:TouchShape =>r|
r:TouchPathBoundingBox =>r | r:TouchDirection =>r | r:PerpendicularTo => r |
r:AngleBetween => r | r:TouchHand => r | r:Tag => r | r:TouchRelativePosition=> r;

          /* AngleBetween*/
syntax AngleBetween =  "Angle between" g1:ValidName "and" g2:ValidName ":" x:ValidNum =>
       AngleBetween {Gesture1=>g1, Gesture2=>g2, Min=>x}
|"Angle between" g1:ValidName "and" g2:ValidName ":" x:ValidNum ".." y:ValidNum =>
       AngleBetween {Gesture1=>g1, Gesture2=>g2, Min=>x, Max=>y};

       /* TouchRelativePosition*/
syntax TouchRelativePosition =  "Relative position between" g1:ValidName "and"
g2:ValidName ":" p:ValidName => TouchRelativePosition {Gesture1=>g1, Gesture2=>g2,
Position=>p};

           /* TouchHand*/
syntax TouchHand = "Touch blob" ":" k:ValidName => TouchHand{Kind=>k}
| "Touch blob" ":" k:ValidName "," t:ValidName => TouchHand {Kind=>k,Type=>t}
| "Touch blob" ":" k:ValidName "," t:ValidName "," s:ValidName => TouchHand {Kind=>k,
Type=>t,Side=>s};

       /* TouchTag*/
 syntax Tag =  "Touch tag" ":" x:ValidNum => Tag {Value=>x};

        /* Perpendicular to*/
syntax PerpendicularTo = g1:ValidName "perpendicularTo" g2:ValidName
    => PerpendicularTo {Gesture1=>g1, Gesture2=>g2};

        /* Touch shape */
syntax TouchShape = "Touch shape" ":" s:Shape =>TouchShape{Values=>s};
token Shape = x: "Line" => x | x: "Rect" => x | x: "Check" => x | x: "Circle" => x;

        /* Touch direction */
syntax TouchDirection = "Touch direction" ":" d:DirectionOptions
            =>TouchDirection{Values=>d};

token DirectionOptions = x: "Left" =>x | x: "Up" =>x | x: "Down" => x| x: "Right" => x
     | x: "UpLeft" => x | x: "UpRight" => x | x: "DownLeft" => x  | x: "DownRight" => x;

syntax DistanceBetweenPoints_Behaviour = "Distance between points" ":"
x:DistanceBetweenPointsOptions => DistanceBetweenPoints{Behaviour=>x};
token DistanceBetweenPointsOptions
= x: "increasing" => x  | x: "decreasing" => x;

        /* Rule: Closed loop */
syntax ClosedLoop = "Closed loop" ":" t:Boolean => ClosedLoop{State=>t};

        /* Rule: Touch Limit */
syntax TouchLimitRule = r:FixedValue => r | r:Range => r;
syntax Range = "Touch limit" ":" x:ValidNum ".." y:ValidNum
            =>TouchLimit{Type=>"Range", Min=>x, Max=>y};

syntax FixedValue = "Touch limit" ":" x:ValidNum
            =>TouchLimit{Type=>"Fixed", Min=>x};

        /* Rule: Touch Step */
syntax TouchStepRule = "Touch step" ":" x:ValidNum "touches" "within"
y:ValidNum z:TouchStepUnitsForTime => TouchStep{TouchCount=>x, TimeLimit=>y, Unit=>z};
token TouchStepUnitsForTime  = "sec" | "msec";

        /* Touch Area */
syntax TouchAreaRule = t:CircularArea => t | t:RectangleArea => t
            | t:EllipseArea => t;

syntax EllipseArea = "Touch area" ":" "Ellipse" x:ValidNum "x" y:ValidNum =>
TouchArea{Type=>"Ellipse", Value=>x+"x"+y};

syntax RectangleAre = "Touch area" ":" "Rect" x:ValidNum "x" y:ValidNum =>
TouchArea{Type=>"Rect", Value=>x+"x"+y}
| "Touch area" ":" "Rect" x:ValidNum "x" y:ValidNum "including" "last" h:ValidNum "touch"
"within" time:ValidNum "sec"=> TouchArea{Type=>"Rect", Value=>x+"x"+y, HistoryLevel=>h,
HistoryTimeLine=>time};

syntax CircularArea = "TouchArea" ":" "Circle" x:ValidNum => TouchArea{Type=>"Circle",
Value=>x};

       /* Touch Time */
syntax TouchTime = "Touch time" ":" val:ValidNum unit:TouchTimeUnit
            =>TouchTime{Value=>val, Unit=>unit };
token TouchTimeUnit  = "sec" | "secs" | "msec" | "msecs";

        /* Touch State */
syntax TouchState = "Touch state" ":" opt:TouchStateOptions =>TouchState{States=>opt};

syntax TouchStateOptions= opt:TouchStateOption+ => opt;
token TouchStateOption = "TouchUp" | "TouchDown" | "TouchMove";

        /* Touch Path Length */
syntax TouchPathLength  = "Touch path length:" min:ValidNum ".." max:ValidNum =>
TouchPathLength{Min=>min, Max=>max}
|"Touch path length:" min:Variable ".." max:Variable => TouchPathLength{VariableMin=>min,
VariableMax=>max}
|"Touch path length:" v:Variable => TouchPathLength{VariableMin=>v}
|"Touch path length:" min:ValidNum  => TouchPathLength{Min=>min};

    }
}
```

9.5 Algorithm that anti-unifies samples of gestures

```
public <List of Primitives> void checkRules()
        {

           // Always create a new solution based on the gestures
            solution = new <List of Primitives>();

            // This loop will search for a relation between the primitives
            Foreach (Sample sample in Samples )
            {
              Foreach  (Primitive in sample)
              {

                if (primitive has more than 1 step)// complex gestures
                {
                     //checks for constant or values that vary inside the established
                     // threshold in primitives that have a numeric value
                     // e.g.: AngleBetween 90
                    checkValueComplexPrimitives(primitive);

                     //checks for constant string in primitives
                     checkStringComplexPrimitives(primitive);
                }
                else
                {
                     //Goes to all the primitives with numeric values
                     //and checks to see if they vary inside a threshold
                     //or if there is a multiplication relationship between them
                    checkProportionforPrimitive(primitive, i);
                     //checks for constant string in primitives
                    checkStringListPrimitives(primitive, 0);

                }
              }
           }
        }
```

9.6 Acquiring images from the device and attaching to correspondent touch points

```
    //Occurs when a new frame of contact data is available
     void _contactTarget_FrameReceived(object sender, FrameReceivedEventArgs e)
        {      //Update image
               if ((touch is not a finger) &&(touch does not contain a tag)))
                    && (ticksDelta > 300000))
               {
                   getImage(e);
                   oldTimeStamp = now;
               }
            }
        }

        private void getImage(FrameReceivedEventArgs e)
        {
            bool imageAvailable = false;
            imageAvailable = e.TryGetRawImage(Normalized,0, 0, Surface.Width,
                        Surface.Height, out normalizedImage,
                        out normalizedMetrics);

            if (imageAvailable)
            {
                imageAvailable = false;
                GCHandle h = GCHandle.Alloc(normalizedImage, GCHandleType.Pinned);
                IntPtr ptr = h.AddrOfPinnedObject();
                Image imageBitmap = new Image(normalizedMetrics.Width,
                                      normalizedMetrics.Height,
                                      normalizedMetrics.Stride,
                                      Format8bppIndexed,ptr);
                ImageHelper.BinarizeImage(imageBitmap);
                 // Creates a new bitmap in order to avoid  memory leakage
                Image imgClone = new Image(imageBitmap);
                imgClone.Palette = imageBitmap.Palette;
                DateTime now = DateTime.Now;
                // Adds the captured image with a timestamp
                snapshots.Add(now, imgClone);
            }
          }
        }
        // Occurs every time a touch event is updated
        public TouchPoint UpdateActiveTouchPoints(TouchPoint touchPoint)
        { // gets the images captured during the touch
            GetSnapShotsFromTouch(touchPoint);
            Image bitImg = getBetterImage(AddSnapshots(touchPoint), touchPoint);

            if (bitImg != null) // there are hand images for this point
            {
                // Add snapshots of hand images with a binarizing filter
                // and calculates the biggest area (palm of hand) and defect areas
                // which are points outside the biggest area (fingers)
                touchPoint.Snapshot = ImageHelper.ExtractContourAndHull(bitImg);
            }
            return touchPoint;
        }
```

9.7 Identifying palm of hand and fingers from a binary image

```
// Receives an image, filters, binarizes it
// and identifies the biggest area (palm) and its
// defects (fingers)
public static TouchImage ExtractContourAndHull(Image<Bgr, Byte> newImg)
        {
            TouchImage touchImage = new TouchImage();
            touchImage.Image = newImg;
            using (MemStorage storage = new MemStorage())
            {
                Image<Gray, Byte> grayImage = touchImage.Image.Convert<Gray, Byte>();
                Contour<System.Drawing.Point> contours =
grayImage.FindContours(Emgu.CV.CvEnum.CHAIN_APPROX_METHOD.CV_CHAIN_APPROX_SIMPLE,
Emgu.CV.CvEnum.RETR_TYPE.CV_RETR_LIST, storage);
                Contour<System.Drawing.Point> biggestContour = null;
                Double Result1 = 0;  Double Result2 = 0;
                while (contours != null)
                {
                    Result1 = contours.Area;
                    if (Result1 > Result2)
                    {
                        Result2 = Result1;
                        biggestContour = contours;
                    }
                    contours = contours.HNext;
                }

                if (biggestContour != null)
                {
                    Contour<System.Drawing.Point> currentContour =
                     biggestContour.ApproxPoly(biggestContour.Perimeter
                     * 0.0025, storage);
                    biggestContour = currentContour;
                    touchImage.Box = biggestContour.GetMinAreaRect();

                    touchImage.Defects = biggestContour.GetConvexityDefacts(storage,
                     Emgu.CV.CvEnum.ORIENTATION.CV_CLOCKWISE).ToArray();
                }
                storage.Dispose();
            }
            return touchImage;
        }
```

9.8 Analyzing images to identify hand postures

```
// On gesture recognition, validate to see if touchpoints are a hand posture
 public <List of Primitives> ValidateHandPosture(<List of TouchPoints> touchPoints)
        {
           <List of Primitives> rules;
            foreach (TouchPoint point in points)
            {
                if (points is not tag
                     and point is not a finger
                     and point contains images)
                 {

                    Primitive hand;
                    HandType type = getHandType(out _type, out _side,  point);
                    if (_type != "")
                    {
                        hand.Type = _type;
                        hand.Side = _side;
                        rules.Add(hand);
                    }
                }
            }
            return rules;
        }

public static void getHandType(out string _type, out string _side, TouchPoint2 point)
        {
            accuracy = 0.9;
            int fingers = ComputeFingersNum(point.Snapshot, out _side);

            if (fingers >= 4)  { _type = TouchHand.OPEN;}
            else if (fingers == 0 && _side == ""){ _type = ""; }
            else { _type = TouchHand.VERTICAL;}
        }

 private static bool isVertical(Rectangle rect)
        {
            return (rect.Width < rect.Height * 0.45);
        }
```

9.9 Calculating the number of fingers and their position

```
public static int ComputeFingersNum(TouchImage touchImage, out string orientation)
       {
           if (touchImage.Box.MinAreaRect().Width
             * touchImage.Box.MinAreaRect().Height < 12000)
           {
               orientation = "";
               return 0;
           }

           if (isVertical(touchImage.Box.MinAreaRect()))
           {
               orientation = "";
               return 1;
           }
           int fingerNum = 1; orientation = ""; PointF LeftEdge = touchImage.Box.center;
           double lowest = 0;
           if (touchImage.Defects == null) {  return 0;}

           for (int i = 0; i < touchImage.Defects.Count(); i++)
           {
           //Calculates the start, depth and endpoint of each defect (potential finger)
               PointF startPoint = new PointF((float)touchImage.Defects[i].StartPoint.X,
                                  (float)touchImage.Defects[i].StartPoint.Y);
               PointF depthPoint = new PointF((float)touchImage.Defects[i].DepthPoint.X,
                                  (float)touchImage.Defects[i].DepthPoint.Y);
               PointF endPoint = new PointF((float)touchImage.Defects[i].EndPoint.X,
                                 (float)touchImage.Defects[i].EndPoint.Y);
             using (touchImage.Defects[i]) {
               LineSegment2D startDepthLine = new LineSegment2D(StartPoint, DepthPoint);
               LineSegment2D depthEndLine = new LineSegment2D(DepthPoint, EndPoint);
               CircleF startCircle = new CircleF(startPoint, 5f);
               CircleF depthCircle = new CircleF(depthPoint, 5f);
               CircleF endCircle = new CircleF(endPoint, 5f);
             }
             //Detecting the orientation of the hand based on the position of the lowest
             // finger, if the lowest finger (thumb) is to the right, then it is left
             // hand, if the lowest finger is to the left, then it is right hand
             //Custom heuristic based on experiment
               if ((startCircle.Center.Y < touchImage.Box.center.Y
                   || depthCircle.Center.Y < touchImage.Box.center.Y) &&
                    (startCircle.Center.Y < depthCircle.Center.Y)
                   && (Math.Sqrt(Math.Pow(startCircle.Center.X - depthCircle.Center.X,2)
                       + Math.Pow(startCircle.Center.Y - depthCircle.Center.Y, 2)) >
                           touchImage.Box.size.Height / 6.5))
               {
                   fingerNum++;
                   if (startCircle.Center.Y > lowest)
                   {
                       lowest = startCircle.Center.Y;
                       if (startCircle.Center.X < depthCircle.Center.X)
                           orientation = TouchHand.LEFT;
                       else
                           orientation = TouchHand.RIGHT;
                   }
                    return fingerNum;
               }
```

9.10 Semi-structure interview script

Interview questions

1. Name, company that the volunteer works in, role the volunteer has in the company
2. Has the volunteer used prototypes before? As a designer? As a user?
    a. What is his opinion about it?
3. Which prototyping tools have you used?
4. What is his opinion regarding paper prototype x software prototype?
5. Can the designer be more specific with the low-fidelity prototypes used, guide him to mention if they are drag'n drop or sketching tools.
6. Prototyping in a tabletop? Has the participant done it?
7. Ask the volunteer to rate the need of some requirements in a low-fidelity prototype tool for tabletop (Fundamental-5, Not relevant-0)
    a. Copy Page (5-0)
    b. Select Components (5-0)
    c. Thumbnails visualization (5-0)
    d. Drag and drop controls
        i. Radiobutton (5-0)
        ii. TextBox (5-0)
        iii. Image (5-0)
        iv. DropDownList (5-0)
        v. Button (5-0)
    e. Annotation tool
        i. Voice (5-0)
        ii. Marker/Sketch (5-0)
        iii. Typing (5-0)
    f. Drawing with a pen x Drawing with finger
8. Ask his take on drag'n drop x sketching tools. Guide him in explaining in what drag and drop tools are good for and in what sketching tools are good for
9. What about the evaluation of the prototypes? Is an annotation tool necessary? Voice or marker? Both?

9.11 Survey for the evaluations in the wild

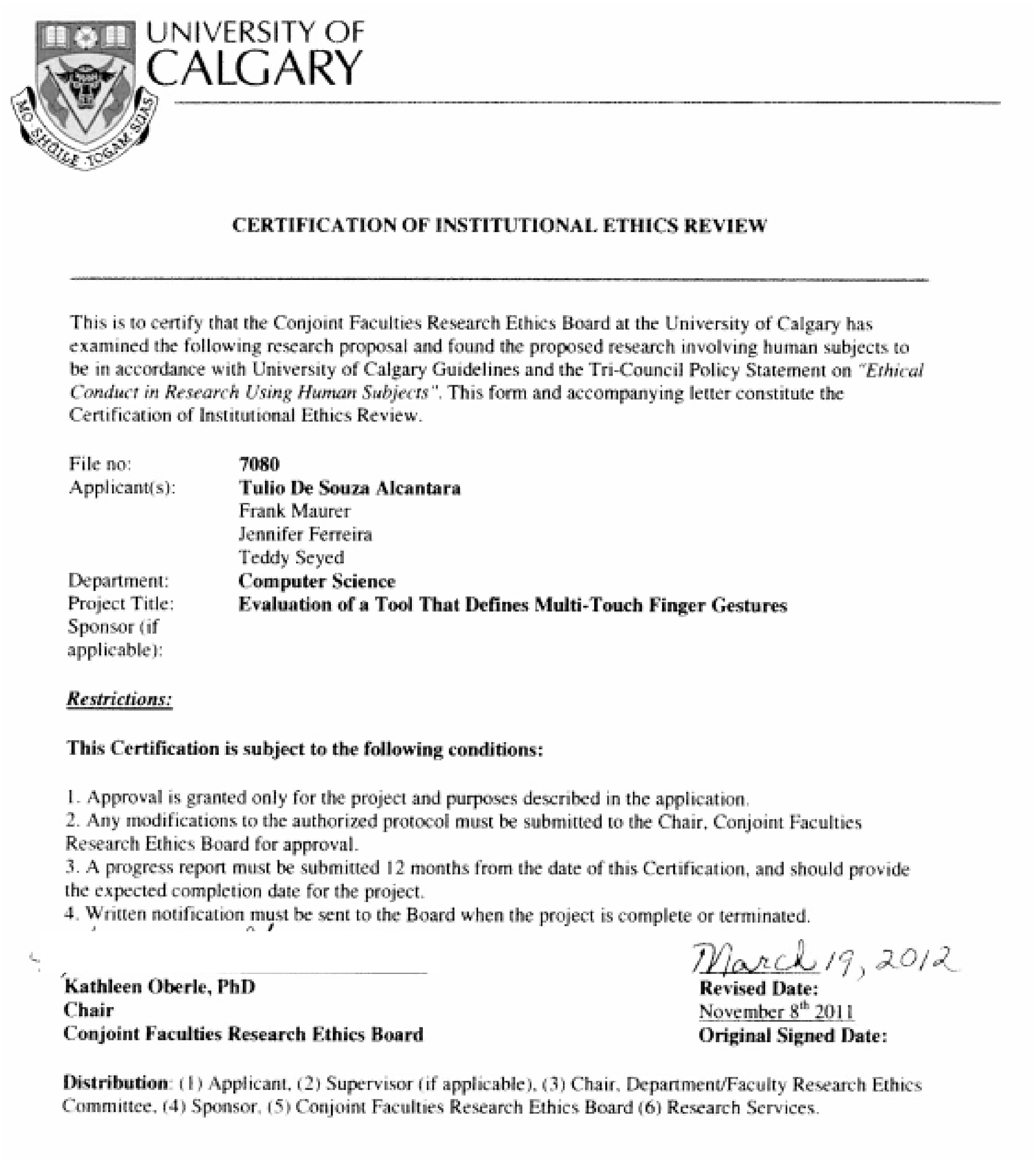
ProtoActive Survey

Name: ________________________________ **Occupation:** ______________________________
(*optional*) (*optional*)

1.	What is your experience in designing touch-based applications?	Choose an item. **Comments:**
2.	What is your experience using low-fidelity prototype?	Choose an item. **Comments:**
3.	What tools have you used for prototyping? (including paper)	
4.	How would you rate the importance of prototyping in designing touch-based applications?	☐Important ☐ If there is time ☐ Not necessary
5.	Have you ever tried to implement (in code) any gestures? If you did, were you satisfied creating gestures in ProtoActive?	☐Yes ☐No
6.	What improvements do you suggest for defining gestures in ProtoActive?	
7.	What improvements do you suggest for ProtoActive?	

8.	Please, list the devices in which you used ProtoActive	
	Device	**Specification**
9.	Please, list the devices in which the final application will be used	
	Device	**Specification**
10.	Please give a brief description of the application you are using ProtoActive to prototype	
11.	Please rate your experience with ProtoActive	Choose an item. **Comments:**
	Additional comments and suggestions	

9.12 Certification of institutional ethics review



UNIVERSITY OF CALGARY

CERTIFICATION OF INSTITUTIONAL ETHICS REVIEW

This is to certify that the Conjoint Faculties Research Ethics Board at the University of Calgary has examined the following research proposal and found the proposed research involving human subjects to be in accordance with University of Calgary Guidelines and the Tri-Council Policy Statement on *"Ethical Conduct in Research Using Human Subjects"*. This form and accompanying letter constitute the Certification of Institutional Ethics Review.

File no:	**7080**
Applicant(s):	**Tulio De Souza Alcantara**
	Frank Maurer
	Jennifer Ferreira
	Teddy Seyed
Department:	**Computer Science**
Project Title:	**Evaluation of a Tool That Defines Multi-Touch Finger Gestures**
Sponsor (if applicable):	

Restrictions:

This Certification is subject to the following conditions:

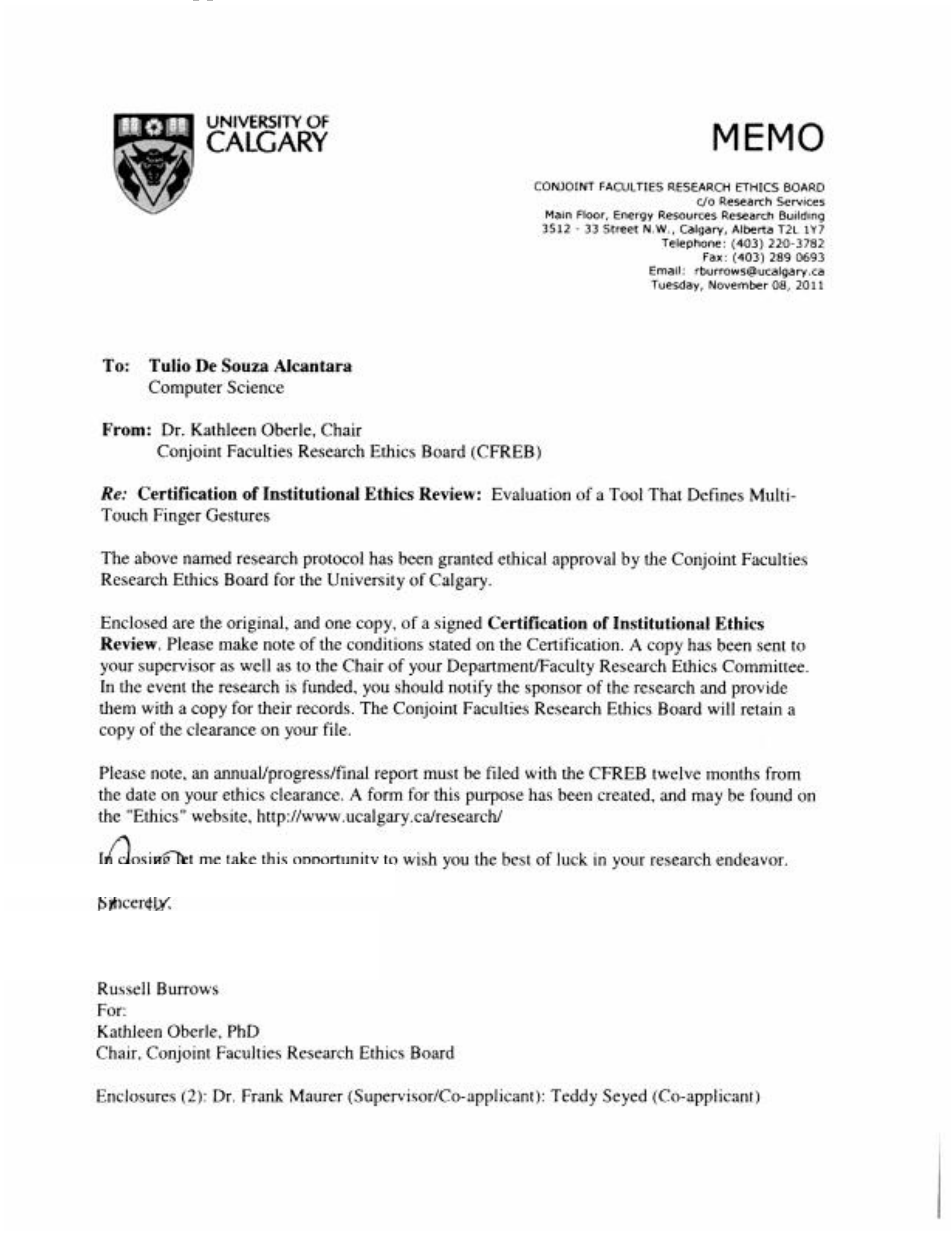
1. Approval is granted only for the project and purposes described in the application.
2. Any modifications to the authorized protocol must be submitted to the Chair, Conjoint Faculties Research Ethics Board for approval.
3. A progress report must be submitted 12 months from the date of this Certification, and should provide the expected completion date for the project.
4. Written notification must be sent to the Board when the project is complete or terminated.

Kathleen Oberle, PhD
Chair
Conjoint Faculties Research Ethics Board

March 19, 2012
Revised Date:
November 8th 2011
Original Signed Date:

Distribution: (1) Applicant, (2) Supervisor (if applicable), (3) Chair, Department/Faculty Research Ethics Committee, (4) Sponsor, (5) Conjoint Faculties Research Ethics Board (6) Research Services.

9.13 Ethics approval



UNIVERSITY OF CALGARY

MEMO

CONJOINT FACULTIES RESEARCH ETHICS BOARD
c/o Research Services
Main Floor, Energy Resources Research Building
3512 - 33 Street N.W., Calgary, Alberta T2L 1Y7
Telephone: (403) 220-3782
Fax: (403) 289 0693
Email: rburrows@ucalgary.ca
Tuesday, November 08, 2011

To: Tulio De Souza Alcantara
Computer Science

From: Dr. Kathleen Oberle, Chair
Conjoint Faculties Research Ethics Board (CFREB)

***Re:* Certification of Institutional Ethics Review:** Evaluation of a Tool That Defines Multi-Touch Finger Gestures

The above named research protocol has been granted ethical approval by the Conjoint Faculties Research Ethics Board for the University of Calgary.

Enclosed are the original, and one copy, of a signed **Certification of Institutional Ethics Review**. Please make note of the conditions stated on the Certification. A copy has been sent to your supervisor as well as to the Chair of your Department/Faculty Research Ethics Committee. In the event the research is funded, you should notify the sponsor of the research and provide them with a copy for their records. The Conjoint Faculties Research Ethics Board will retain a copy of the clearance on your file.

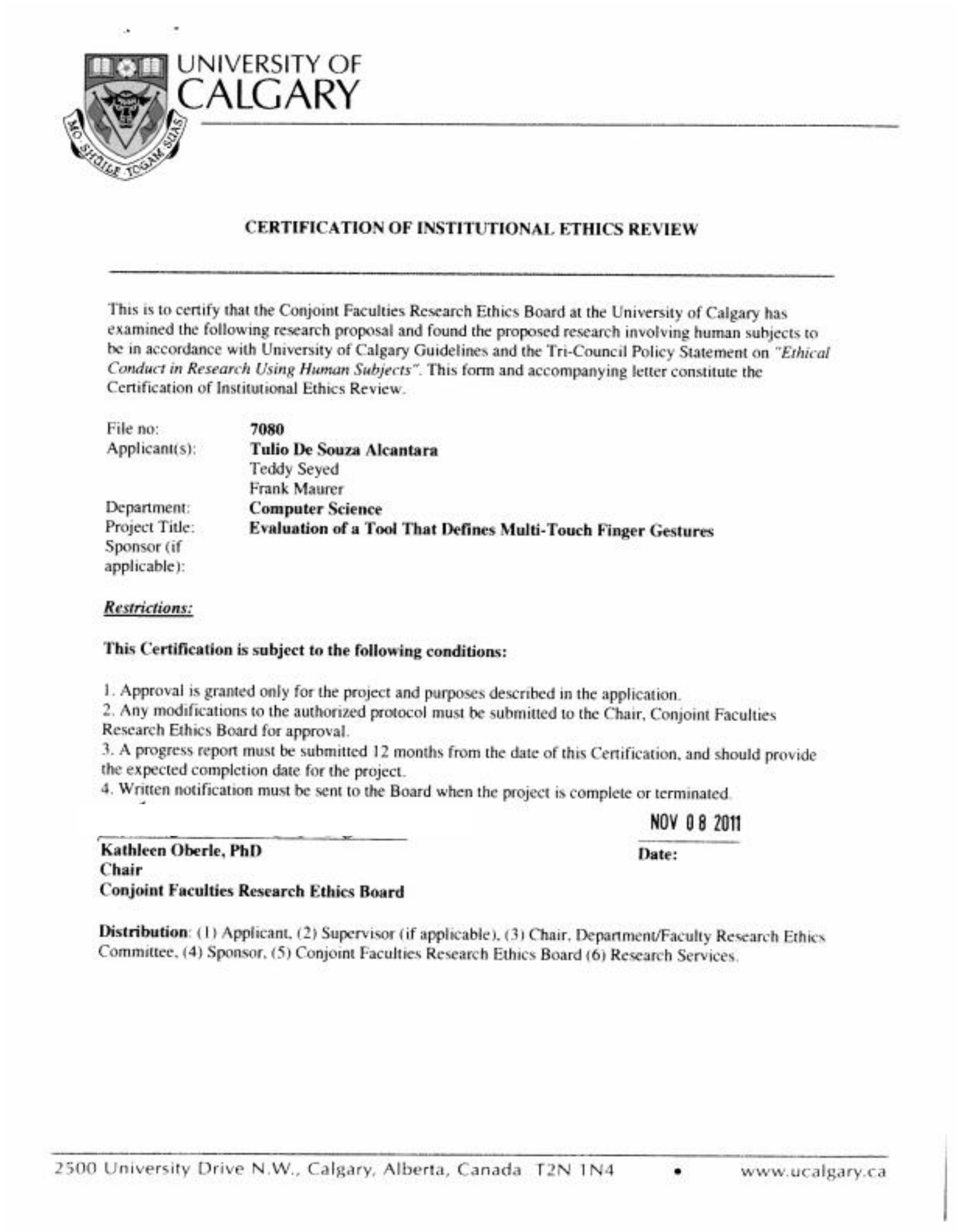
Please note, an annual/progress/final report must be filed with the CFREB twelve months from the date on your ethics clearance. A form for this purpose has been created, and may be found on the "Ethics" website, http://www.ucalgary.ca/research/

In closing let me take this opportunity to wish you the best of luck in your research endeavor.

Sincerely,

Russell Burrows
For:
Kathleen Oberle, PhD
Chair, Conjoint Faculties Research Ethics Board

Enclosures (2): Dr. Frank Maurer (Supervisor/Co-applicant): Teddy Seyed (Co-applicant)



UNIVERSITY OF CALGARY

CERTIFICATION OF INSTITUTIONAL ETHICS REVIEW

This is to certify that the Conjoint Faculties Research Ethics Board at the University of Calgary has examined the following research proposal and found the proposed research involving human subjects to be in accordance with University of Calgary Guidelines and the Tri-Council Policy Statement on *"Ethical Conduct in Research Using Human Subjects"*. This form and accompanying letter constitute the Certification of Institutional Ethics Review.

File no:	**7080**
Applicant(s):	**Tulio De Souza Alcantara** Teddy Seyed Frank Maurer
Department:	**Computer Science**
Project Title:	**Evaluation of a Tool That Defines Multi-Touch Finger Gestures**
Sponsor (if applicable):	

Restrictions:

This Certification is subject to the following conditions:

1. Approval is granted only for the project and purposes described in the application.
2. Any modifications to the authorized protocol must be submitted to the Chair, Conjoint Faculties Research Ethics Board for approval.
3. A progress report must be submitted 12 months from the date of this Certification, and should provide the expected completion date for the project.
4. Written notification must be sent to the Board when the project is complete or terminated.

NOV 08 2011

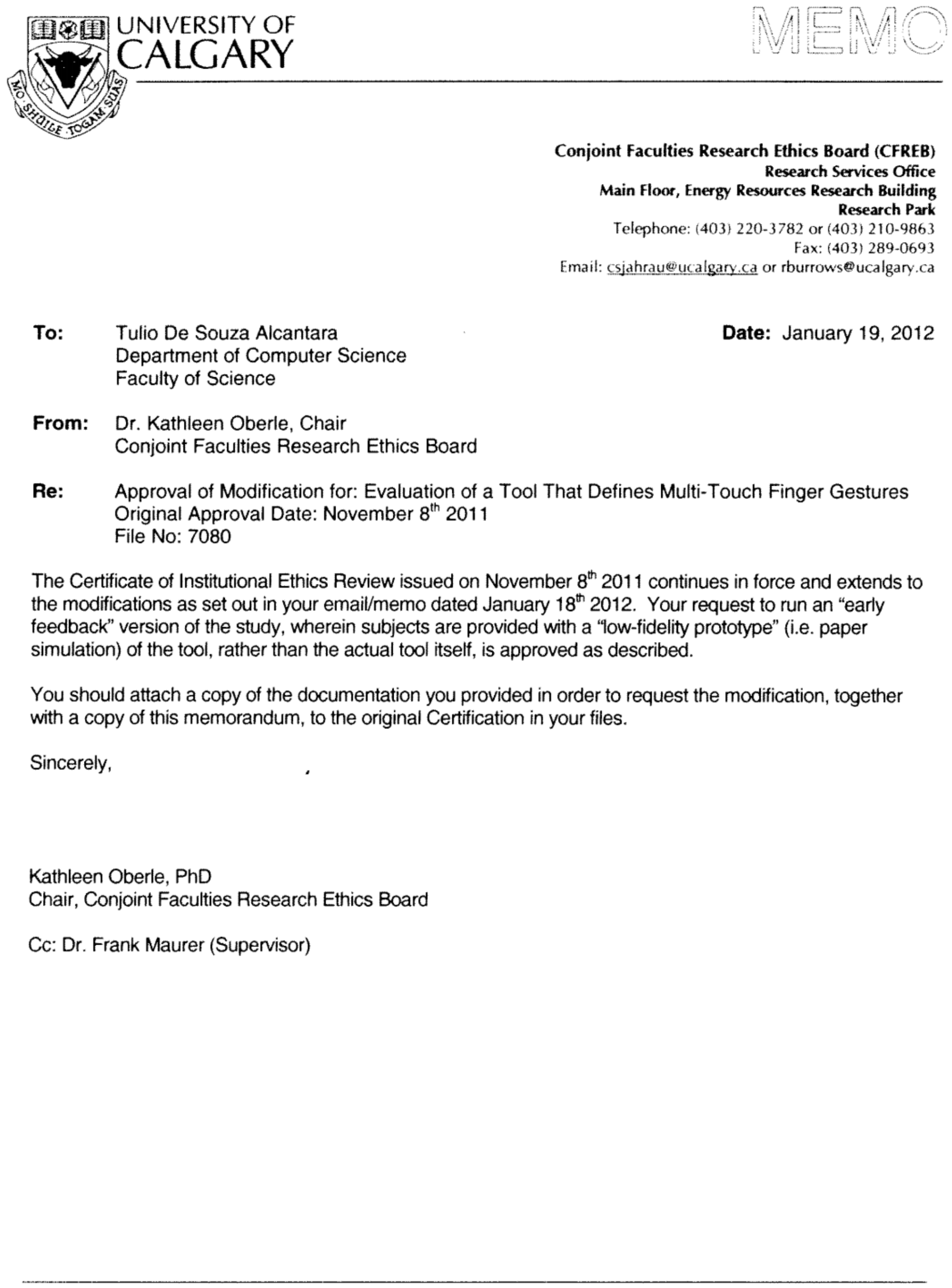
Kathleen Oberle, PhD
Chair
Conjoint Faculties Research Ethics Board

Date:

Distribution: (1) Applicant, (2) Supervisor (if applicable), (3) Chair, Department/Faculty Research Ethics Committee, (4) Sponsor, (5) Conjoint Faculties Research Ethics Board (6) Research Services.

2500 University Drive N.W., Calgary, Alberta, Canada T2N 1N4 • www.ucalgary.ca

9.14 Ethics approval extension #1



UNIVERSITY OF CALGARY

MEMO

Conjoint Faculties Research Ethics Board (CFREB)
Research Services Office
Main Floor, Energy Resources Research Building
Research Park
Telephone: (403) 220-3782 or (403) 210-9863
Fax: (403) 289-0693
Email: csjahrau@ucalgary.ca or rburrows@ucalgary.ca

To: Tulio De Souza Alcantara
Department of Computer Science
Faculty of Science

Date: January 19, 2012

From: Dr. Kathleen Oberle, Chair
Conjoint Faculties Research Ethics Board

Re: Approval of Modification for: Evaluation of a Tool That Defines Multi-Touch Finger Gestures
Original Approval Date: November 8th 2011
File No: 7080

The Certificate of Institutional Ethics Review issued on November 8th 2011 continues in force and extends to the modifications as set out in your email/memo dated January 18th 2012. Your request to run an "early feedback" version of the study, wherein subjects are provided with a "low-fidelity prototype" (i.e. paper simulation) of the tool, rather than the actual tool itself, is approved as described.

You should attach a copy of the documentation you provided in order to request the modification, together with a copy of this memorandum, to the original Certification in your files.

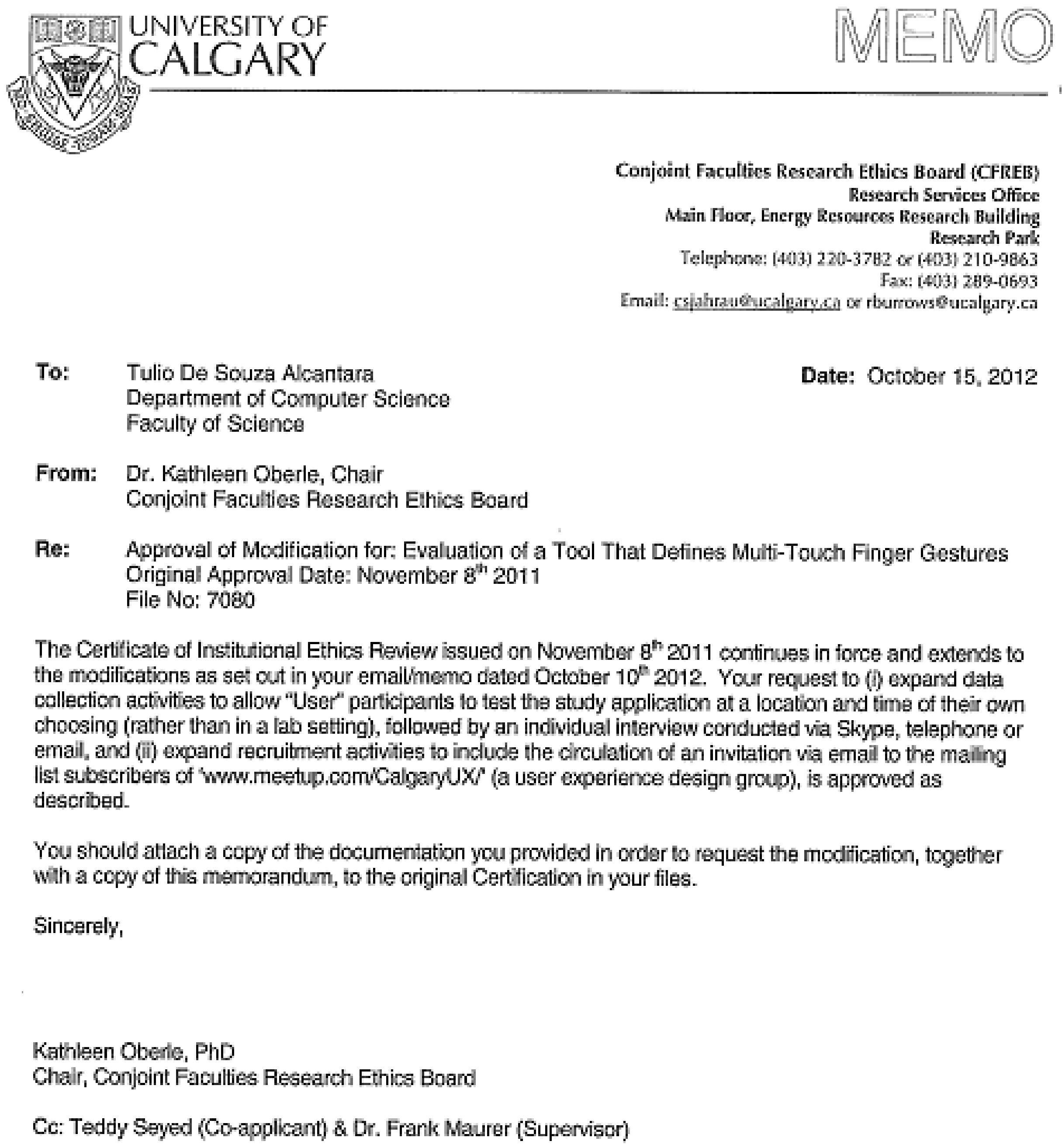
Sincerely,

Kathleen Oberle, PhD
Chair, Conjoint Faculties Research Ethics Board

Cc: Dr. Frank Maurer (Supervisor)

2500 University Drive N.W., Calgary, Alberta, Canada T2N 1N4 • www.ucalgary.ca

9.15 Ethics approval extension #2



UNIVERSITY OF CALGARY

MEMO

Conjoint Faculties Research Ethics Board (CFREB)
Research Services Office
Main Floor, Energy Resources Research Building
Research Park
Telephone: (403) 220-3782 or (403) 210-9863
Fax: (403) 289-0693
Email: csjahrau@ucalgary.ca or rburrows@ucalgary.ca

To: Tulio De Souza Alcantara
Department of Computer Science
Faculty of Science

Date: October 15, 2012

From: Dr. Kathleen Oberle, Chair
Conjoint Faculties Research Ethics Board

Re: Approval of Modification for: Evaluation of a Tool That Defines Multi-Touch Finger Gestures
Original Approval Date: November 8th 2011
File No: 7080

The Certificate of Institutional Ethics Review issued on November 8th 2011 continues in force and extends to the modifications as set out in your email/memo dated October 10th 2012. Your request to (i) expand data collection activities to allow "User" participants to test the study application at a location and time of their own choosing (rather than in a lab setting), followed by an individual interview conducted via Skype, telephone or email, and (ii) expand recruitment activities to include the circulation of an invitation via email to the mailing list subscribers of 'www.meetup.com/CalgaryUX/' (a user experience design group), is approved as described.

You should attach a copy of the documentation you provided in order to request the modification, together with a copy of this memorandum, to the original Certification in your files.

Sincerely,

Kathleen Oberle, PhD
Chair, Conjoint Faculties Research Ethics Board

Cc: Teddy Seyed (Co-applicant) & Dr. Frank Maurer (Supervisor)

9.16 Consent form

Name of Researcher, Faculty, Department, Telephone & Email:

Túlio Alcantara – Graduate Student Department of Computer Science E-mail: tuliosouza@gmail.com Phone: (403) 210-9499	Jennifer Ferreira – Posdoctoral fellow Department of Computer Science E-mail: jen.ferreira@ucalgary.ca Phone: (403) 210-9499	Frank Maurer – Professor Department of Computer Science E-mail: frank.maurer@ucalgary.ca Phone: (403) 220-3531

Title of Project:

A Study of touch based gestures in Software Development and Interaction Design

This consent form, a copy of which has been given to you, is only part of the process of informed consent. If you want more details about something mentioned here, or information not included here, you should feel free to ask. Please take the time to read this carefully and to understand any accompanying information.

The University of Calgary Conjoint Faculties Research Ethics Board has approved this research study.

Purpose of the Study:

The purpose of this study is to help researches evaluate a tool that defines multi-touch finger gestures. We are interested in evaluate the performance of our, as well as get user feedback in ways the tool can be improved

What Will I Be Asked To Do?

You will be provided with the tool via email or website and an explanation of the tool's usage. With the tool in your possession, researchers will ask for incremental feedback.

After the experiment, you will be asked to participate in an interview or answer a survey about your experience with the tool and/or your experience with similar tools.

Participation is absolutely voluntary and you may withdraw at any time.

What Type of Personal Information Will Be Collected?

No personal identifying information will be asked as well as no identify information about previous and/or current jobs. All participants shall remain anonymous.

Are there Risks or Benefits if I Participate?

There are no known harms or risks associated to the participation in this study.

What Happens to the Information I Provide?

This information will be kept in a secure location (password-protected discs). The information that we collect will not be associated to you personally, however, the researchers will publish the results of their analysis of your data in anonymized form in academic journals and conferences.

The researchers might quote the responses in the interview or any of your comments in anonymized form.

All the collected data will be kept by the investigators indefinitely and it might be used with publications and thesis purposes.

Signatures (written consent)

Select either box gives us consent to (X to indicate choice):

	(agree)	(not agree)
Save prototype data	______	______
Save Interview data	______	______
Save Survey data	______	______

Your signature on this form indicates that you 1) understand to your satisfaction the information provided to you about your participation in this research project, and 2) agree to participate as a research subject.

In no way does this waive your legal rights nor release the investigators, sponsors, or involved institutions from their legal and professional responsibilities. You are free to withdraw from this research project at any time. You should feel free to ask for clarification or new information throughout your participation.

Participant's Name: (please print) ______________________________

Participant's Signature ______________________________ Date: ______________

Researcher's Name: (please print) TÚLIO DE SOUZA ALCANTARA

Researcher's Signature: Date: 30 OCTOBER 2017

Questions/Concerns

If you have any further questions or want clarification regarding this research and/or your participation, please contact:

Túlio Souza
Department of Computer Science
(403) 210-9499, tdsalcan@ucalgary.ca
Or
Jennifer Ferreira
Department of Computer Science
(403) 210-9499, jen.ferreira@ucalgary.ca

Or
Frank Maurer
Department of Computer Science
(403) 220-3531, frank.maurer@ucalgary.ca

If you have any concerns about the way you've been treated as a participant, please contact the Senior Ethics Resource Officer, Research Services Office, University of Calgary at (403) 220-3782; email rburrows@ucalgary.ca.

A copy of this consent form has been given to you to keep for your records and reference. The investigator has kept a copy of the consent form.