FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Multi-touch Interaction for Interface Prototyping

João Pedro Pereira da Costa Portela

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Rui Pedro Amaral Rodrigues (PhD)

July 13, 2012

The work done on this dissertation was partially supported by the BIC grant: PTDC/EIA-EIA/108982/2008

Multi-touch Interaction for Interface Prototyping

João Pedro Pereira da Costa Portela

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Doctor Jorge Alves da Silva
External Examiner: Doctor Pedro Miguel do Vale Moreira
Supervisor: Doctor Rui Pedro Amaral Rodrigues

July 13, 2012

Abstract

Changes on interaction technologies over the last decades have the potential to mark a paradigm shift from Graphical User Interfaces (GUI), which has been the dominant interaction paradigm for thirty years. These new devices aim to provide a more natural way of interaction, constituting the Natural User Interfaces (NUI), by taking advantage of the skills we have naturally acquired through our experience with the world.

The success of multi-touch displays, one of the interfaces that benefit from a NUI design, made designers rush into developing novel interfaces for them. However, they end up ignoring and violating well-known interaction design principles. In order to find standards in designing these, designers should be allowed the freedom to experiment. Prototyping is recommended, in order to reduce the cost of experimenting. However, the tool support is limited in this area.

At the same time, prototyping tools can benefit from the NUI paradigm, in particular by running on a multi-touch display, which is something that is still lacking research and exploration. These interfaces offer more versatile and natural way of prototyping, similar to paper-based prototyping, which is the method preferred by interaction designers.

This dissertation addresses these problems by proposing a prototyping tool that runs on a multi-touch tabletop, leveraging the NUI paradigm, and that allows the creation of prototypes suitable for both GUI and multi-touch interfaces. The solution was specified in terms of functionality and an approach to implement it, based on User-Centered Design, was followed.

An aim of this tool is to be generic enough to prototype novel interactions. As such its architecture is based in the Object, Containers, Gestures and Manipulations (OCGM) framework, and implements and extends the Proton Manipulation Framework for Gestures and Manipulations, for flexible manipulation specification. Both the architecture and the implementations details are described and discussed.

A functional prototype, implementing a subset of selected features, was developed and validated through user testing sessions. The insights gathered from these tests helped us validate future work on this solution, which is presented and clearly defined, in order to plan a future iteration of development.

It is expected that the contributions of this work, resulting from applying and extending the Proton and OCGM frameworks, as well as from exploring prototyping in NUI environments, can help and inspire future designers and developers in this area.

Resumo

As mudanças nas tecnologias de interação que têm surgido nas últimas décadas, têm o potencial de mudar o paradigma de interação atual, as Interfaces Gráficas (GUI), dominante há trinta anos. Estes novos aparelhos têm o objetivo de oferecer uma forma mais natural de interação, constituindo as Interfaces Naturais (NUI). Estas interfaces tentam utilizar as capacidades que adquirimos naturalmente, ao longo da nossa experiência com o mundo.

O sucesso dos ecrãs multi-toque, uma das interfaces naturais, fez com que os *designers* se apressassem em desenvolver novas interfaces para os mesmos. No entanto, estes acabam por ignorar e violar princípios de *design* de interação bem estabelecidos. De forma a encontrar padrões de *design* para os mesmos, os *designers* devem ter a liberdade de experimentá-los. Neste contexto, a prototipagem é recomendada, de forma a reduzir o custo da experimentação. Existe, no entanto, falta de suporte de ferramentas nesta área.

Ao mesmo tempo, ferramentas de prototipagem podem beneficiar do paradigma NUI, em particular se utilizarmos estas ferramentas num ambiente de ecrã multi-toque, o que é algo que ainda falta investigação e exploração. Estas interfaces oferecem uma forma mais versátil e natural de prototipagem, semelhante à forma de prototipar em papel, que é o método preferido pelos *designers* de interação.

Esta dissertação dirige-se a estes problemas, ao propor uma ferramenta de prototipagem, que funciona numa mesa multi-toque, aproveitando o paradigma de NUI, e que permite a criação de protótipos tanto para GUI como para interfaces multi-toque. A solução foi especificada em termos de funcionalidade e de abordagem para a sua implementação, baseada em *Design* Centrado no Utilizador.

Um dos objetivos desta ferramenta é que seja genérica o suficiente para prototipar novas interações. Como tal, a arquitectura proposta é baseada na *framework* de Objetos, Contentores, Gestos e Manipulações (OCGM), implementando a *Framework* de Manipulações Proton para Gestos e Manipulações. A arquitetura e alguns detalhes da implementação são descritos e discutidos.

Um protótipo funcional que implementa um subconjunto de funcionalidades foi desenvolvido e validado, através de sessões de testes com utilizadores. Várias ideias e perspetivas foram retiradas através da análise dos resultados destes testes, ajudando a validar qual deverá ser o trabalho futuro sobre esta solução, sendo este apresentado e claramente definido, de forma a planear uma iteração futura de desenvolvimento.

Várias contribuições foram feitas que poderão ser úteis para a comunidade académica e para a indústria, através da aplicação e extensão das *frameworks* Proton e OCGM, assim como através da exploração da prototipagem em ambientes NUI, que poderá inspirar *designers* e *developers* no futuro, a trabalhar nesta área.

Acknowledgements

One page is not enough to acknowledge everyone who has contributed to this dissertation, the culmination of five years of my academic life. I know those will understand.

My first words are directed to my supervisor, Prof. Rui Rodrigues, because this dissertation would not exist without him. The initial idea of a prototyping tool running on a multi-touch platform, along with some concepts of interaction, came from him. It was a great pleasure to take this idea and develop it into a concrete project. His direct contribution is unquestionable, and I want to thank him for exposing me to Interaction Design and Natural User Interfaces. I also want to thank the seven testers, who directly helped enrich this dissertation.

The final, longer words are for those who inevitably shaped what I and this dissertation have become. For those, the words are in the language that better translates what I feel.

Em primeiro lugar, umas palavras de agradecimento à minha família, por me terem educado e apoiado sempre, na concretização dos meus sonhos. À minha mãe, Goreti, pela mão firme em educar, e flexível em ajudar a encontrar as minhas paixões. Ao meu pai, Benjamim, pela enorme paciência, sacrifício por acompanhar os filhos e por todas as boleias. À minha irmã Diana, por ser um modelo a seguir, principalmente na faculdade, e ao meu irmão Pedro por me obrigar todos os dias a ser um modelo para ele. Ao meu cunhado Pedro, pelas tardes e noites a jogar e conversar. À minha avó Lurdes, por me acompanhar constantemente, sem falhar, e aos meus tios Lídia e Barnabé por me terem sempre acolhido na sua mesa.

Um grande abraço e agradecimento aos meus amigos de S. Pedro da Cova que me acompanharam até ao 12º ano - e que me continuam a acompanhar fora da faculdade. Ao Ricardo, pela inteligência e capacidade de trabalho que me inspiraram, e pela eterna amizade. Ao Ivo, pela boa disposição, espírito de dedicação e humildade. Ao Josué, pela força, humildade e espirito brincalhão. Ao Pedro, ao Dani, à Daniela, aos dois Diogos, à Rita e aos restantes membros da TJ 27, por me acolherem sempre com entusiasmo e boa disposição, fora da vida de informático.

Finalmente, as palavras mais pessoais vão para quem me acompanhou e me manteve motivado, não só durante esta dissertação, mas durante uma vida de 5 anos, que será irrepetível. Ao Oleksandr, ao Rui e ao Daniel, pelas horas incontáveis passadas no NIFEUP a ouvir-me falar da minha tese. Ao Lucas, pelas conversas filosóficas e pela boa disposição. Ao Fábio, pelas boleias para casa e pelo esforço contagiável, aplicado em todos os trabalhos. Ao Diogo, pelo companheirismo indiscutível e amor ao clube. À Maria, pela boa disposição contagiável. À Sara, pelo apoio e amizade durante este ano, e por me ter feito crescer imenso como pessoa. À Sónia, por ser a melhor líder, pela capacidade de organização de motivar os outros a superarem-se. Ao Eduardo, pela incrível dedicação, por ser um modelo a seguir e por me ter desafiado em subir a minha média. Por último, à Beatriz, que me acompanhou e deu força nos últimos momentos desta enorme jornada. Aos restantes, vos saúdo.

João Portela

"It's kind of fun to do the impossible."

Walt Disney

Contents

List of Figures

List of Tables

Abbreviations

AST	Abstract Syntax Tree
BNF	Backus Naur Form
CLI	Command-Line Interfaces
DI	Diffused Illumination
DSI	Diffused Surface Illumination
FTIR	Frustrated Total Internal Reflection
GUI	Graphical User Interfaces
LALR	Look-Ahead Left Right
LASER	Light Amplification by Stimulated Emission of Radiation
LED	Light-Emitting Diode
LED-LP	LED Light Plane
LLP	LASER Light Plane
MPX	Multi-Pointer X
NUI	Natural User Interfaces
OCGM	Objects, Containers, Gestures and Manipulations
PARC	Palo Alto Research Center
UCD	User-Centered Design
UI	User Interface
US	User Story
WIMP	Windows, Menus, Icons and Pointer
WYSIWYG	What You See Is What You Get
XML	eXtensible Markup Language

Chapter 1

Introduction

This chapter presents the background and motivation, the problems addressed and the goals of this dissertation. It also presents the structure of this dissertation report.

1.1 Background and Motivation

Computers and everyday technology have evolved at a steep rate in the last decades, both in terms of hardware and software. In spite of these advancements, the way we interact with computers has not changed significantly since the 1980s, when Graphical User Interfaces (GUI) were first introduced. In fact, the "Windows, Icons, Menus, Pointer" (WIMP) interaction style developed in the 1980s is still the basis for today GUIs [RSP11].

There was definitely an evolution to this interaction style in the past decades (mostly in the form of better graphics, supported by better hardware, and more control types, such as toolbars, sliders and docks). However, the stagnancy of the interaction paradigm, which relies on the mouse and the keyboard, is evident. The adequacy of the WIMP to office environments is one of the major factors that made the GUI survive three decades since its inception [WW11, Van97].

In the last two decades, new kinds of interaction technology have been developed. These technologies are built upon the user's knowledge of the real world [JGH+08], having the potential to shift the interaction paradigm from the GUI. They are called the Natural User Interfaces (NUI), with the rationale of taking advantage of the skills we have naturally acquired through our experience with the world [GB10]. NUI principles can be applied in a diverse range of interfaces, such as touch, voice, gesture, tangible and organic based interfaces.

The availability of multi-touch displays (smartphones and tablets) to the masses, caused designers to rush into developing and experimenting new interfaces for this hardware, labeling them as natural. However, most of the times they end up ignoring and violating established interaction design principles [NN10]. In order to find standards in designing NUIs, designers should be allowed the freedom to experiment [NN10]. Given the expensive resources involved in developing

a software product and the competitive market, it is common to use low-fidelity prototypes, which encourage creativity, while gathering user feedback [KJ06, Pet07]. Tool support specifically designed for creating NUI prototypes is currently quite limited[HKSBM11], apart from paper-based prototypes which are sufficiently versatile to prototype these interfaces [War09].

Prototyping tools are also a good candidate for benefiting from the NUI paradigm, in particular by running on a multi-touch display, which is an approach still lacking research and exploration. Multi-touch can wield a higher bandwidth input from the user. Tabletops can help engaging users in a more intuitive and direct way of manipulation and encourage multi-user collaboration [WW11, KAD09, FWSB07]. This makes this interface a candidate for creating a more versatile and natural way of prototyping, similar to paper-based prototyping, which is the method preferred by most interaction designers [CH10, War09].

This dissertation addresses the Human-Computer Interaction area, under the following topics: Interaction Design, User Interface Prototyping, Natural User Interfaces and Multi-touch Interaction.

1.2 Problem Definition

This dissertation addresses two problems:

1. Paper based prototyping tools are more used by interaction design practitioners than software based ones [CH10, War09]. The reason for this seems to be the versatility and natural way of manipulating paper, that provides an immediate way of translating thought into paper. There is an opportunity to apply Natural User Interfaces principles to create a new interface prototyping tool, which is an approach still lacking research.

2. There is a lack of tool support for prototyping multi-touch interaction. Given the need to experiment these interfaces, addressing this problem is a very important step towards establishing guidelines and standards for this area.

1.3 Goals and Methodology

This dissertation has the goal of creating the foundations for a prototype design tool. In order to address the problems identified above, the tool should:

1. take advantage of a multi-touch display, leveraging the NUI paradigm.

2. create prototypes suited for multi-touch displays and NUI interaction.

To better illustrate these goals, Figure 1.1 depicts the current scenario on User Interface Prototyping. There are software tools, based on GUI/WIMP interaction, where designers can create prototypes for GUI/WIMP interfaces.

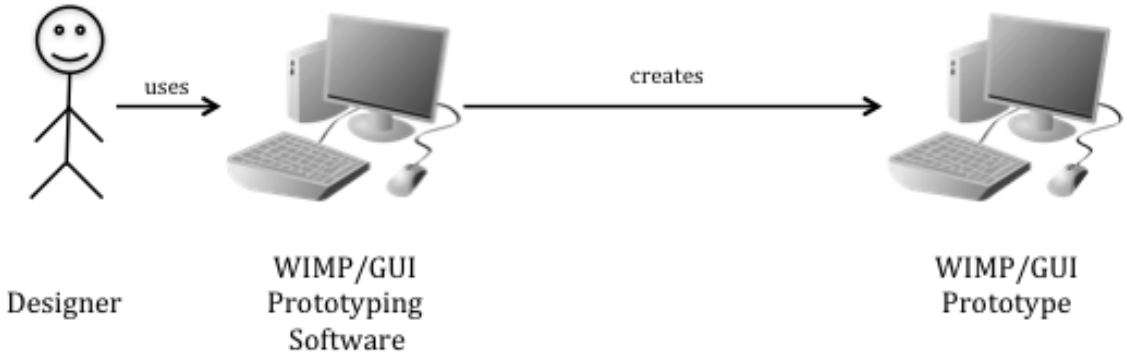

Figure 1.1: Current scenario of interface prototyping tools

Figure 1.2 shows the scenario this dissertation envisions: a software tool, based on Multi-touch/NUI interaction, where designers can create prototypes for GUI/WIMP interfaces and Multi-touch/NUI interfaces.

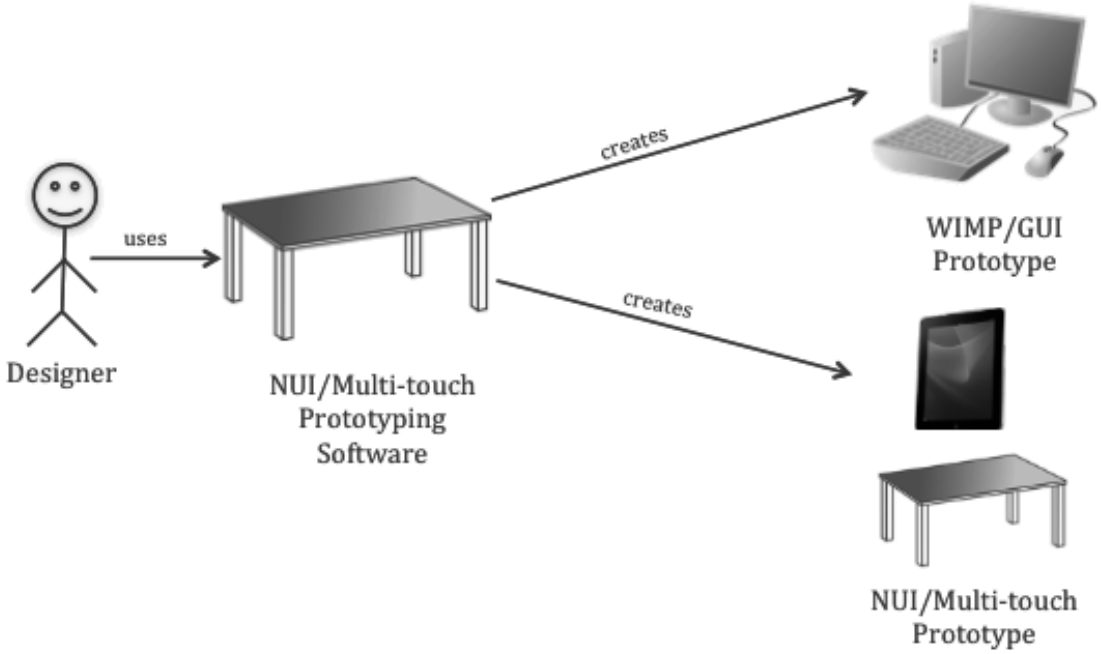

Figure 1.2: The proposed (NUI) scenario of interface prototyping tools

In the context of this work, the following steps were carried to accomplish these goals:

1. An overview of the Human-Computer Interaction field was done, to gain insight in how this field has evolved and how it might change.

2. Interaction Design principles and guidelines were gathered, which should be used as a basis for the design of Natural User Interfaces.

3. The practice of interface prototyping was researched and the available prototyping tools were compared, in order to find what functionality the solution should cover.
4. Natural User Interfaces and their principles were studied, in order to understand the grounds in which the solution will be designed.
5. A more technical study on multi-touch interfaces was done, addressing the platform limitations and design guidelines.
6. A solution proposal, in terms of functionality and approach, was conceived.
7. An architecture design to support the development of the tool and support future changes was specified.
8. A functional prototype for the solution was implemented, using a milestone-based approach for development.
9. User testing sessions were conducted, in order to validate the solution and gather insights for future development.

1.4 Document Structure

This document has a total of seven chapters, including this introduction. In Chapter 2, the state of the art of the field that is applicable and useful for this dissertation, is described.

The conceptualization and specification of the solution can be found in Chapter 3, in terms of approach and functionality. Chapter 4 details the architecture created to support the specified solution. Chapter 5 presents important details concerning the implementation of the solution and describes the functional prototype developed.

In Chapter 6, the user tests used to validate the solution are presented, by describing their methodology and form, and by analyzing and discussing their results.

Finally, Chapter 7 presents the conclusions, the contributions of this dissertation to the community and perspectives for future work.

Chapter 2

Review on Interaction Design and Natural User Interfaces

In this chapter the state of the art that is applicable and useful for this dissertation is described.

The Human-Computer Interaction area is introduced in Section 2.1, presenting a historical perspective on the User Interface paradigms and documenting the limitations of the current paradigm, the Graphical User Interface. This sets the tone and inspiration for the work done on this dissertation. Section 2.2 proceeds by presenting the field of Interaction Design and why it is important on the context of designing Natural User Interfaces. Some key principles and guidelines are introduced, as well as established references on the area, and a methodology for User-Centered Design (UCD) is presented. Interface Prototyping, a key technique in UCD is presented in Section 2.3, with a study of the common tools used in this technique.

Section 2.4 presents Natural User Interfaces (NUI), in terms of their foundations and principles and introduces the OCGM interaction style as universal founding metaphors of NUIs. Finally, in section 2.5, multi-touch technology is introduced, by reviewing its hardware (and the limitations of each technique), as well as a comparative study of multi-touch frameworks available.

2.1 Human-Computer Interaction

Human-computer interaction is a field that spans a wide range of disciplines, not only computer science, but also psychology, sociology and anthropology. The definition provided by the ACM SIGCHI Curricula for Human-Computer Interaction is [HBC+92]:

> "Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them."

The study of the interaction between humans (the users) and computers has gained more relevance in recent years. The amount of new technological developments has encouraged different ways of thinking about interaction design [RSP11].

The way the user accomplishes tasks with a system, what the user does and what the system responds is the *interface*. The importance of having a good interface cannot be stressed enough, and should be one of the most important concerns when developing software. Users do not care about the inner workings of a product, as long as it does what needs to be done. All they touch, feel, see and hear is the interface (Figure 2.1). As far as the user is concerned, the interface is the product [Ras00].

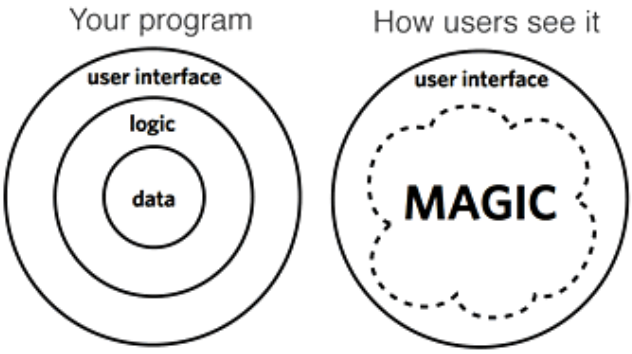

Figure 2.1: The way users see the system [Hei09]

2.1.1 History of User Interfaces

For a better understanding of how user interfaces can evolve in the future, it is interesting to see how they evolved in the past. The history of user interfaces can be characterized as having long periods of stability, interrupted by rapid change [Van97]. Traditionally, there are three main paradigms on the history of user interfaces[1] [RL04]:

- *Batch Interfaces (1945-1968).* The scarce computing power available on early computer systems, made the user interface be considered as an overhead to the system. Users had to accommodate to computers, instead of the other way around. Users did not interact with the system in real-time, having to prepare a deck of punched cards describing a program and a dataset, that would be later processed in batches.

- *Command-Line Interfaces (1969-1983).* Command-Line Interfaces (CLI) had an interaction model based on a series of request-response transactions, with requests expressed as textual commands in a specialized vocabulary. Software could be used in new exploratory and interactive ways. However, these interfaces placed a heavy mnemonic load on the user, requiring a serious investment in learning, in order to master them.

[1] The dates provided correspond to the establishment of the paradigm, in terms of commercial production.

- *Graphical User Interfaces (1984-present).* The Graphical User Interfaces allowed users to interact with the computer visually, by providing intuitive metaphors for representation and manipulation of entities and actions (with the introduction of the mouse and the pointer). GUIs main strength was in how easy it was for users to remember what actions existed and how to invoke them [Nor10].

Other types of interfaces coexist that have not managed to establish themselves as a main user interface paradigm. Examples of these are [WW11, RSP11]: the Menu-Based Interfaces (ATMs), Fun User Interfaces (videogames)[Shn04], Web User Interfaces [Kru06], Pen-Based Interfaces[LMZ05], Gestural Interfaces (Kinect) [VRPS11], Appliance Interfaces (home machines), Augmented Reality Interfaces [ABB+01], Virtual Reality Interfaces and Tangible Interfaces [HB06].

2.1.2 Graphical User Interfaces

The engineering tradition behind today's GUIs was born at the Xerox Palo Alto Research Center (PARC), in the 1970s. The Xerox Star was the first system that brought this technology into the market. Although it was a commercial failure, it introduced the familiar GUI interaction style still in use today: the WIMP (Windows, Icons, Menus and Pointer). This interaction style was popularized by the Apple Macintosh in 1984, with the introduction of the mouse in a personal computer environment [RL04].

Graphical User Interfaces had a considerable impact in the Human-Computer Interaction field. The main strength of the GUI is the ease of remembering actions, both in what actions are possible and how to invoke them. This complies with the visibility design rule: through the menus, all possible actions can be made visible and, therefore, easily discoverable. The system can often be learned through exploration [Nor10]. GUIs also helped lowering the barriers between users and the computer. Plus, they provided considerable functionality gains for most users [WW11].

2.1.2.1 WIMP Interaction Style

The WIMP interaction style describes the set of elements designed for the GUI developed by Xerox PARC in 1980, referring to the Windows, Icons, Menus and Pointer. These elements survived for more than three decades, and still constitute the basic building blocks of today GUIs [RSP11]:

- *Windows*. The metaphor of windows allowed to better organize space on the computer screen, and allowed multiple programs to be opened at the same time.
- *Icons*. The visual representation of applications, objects, commands and tools, that were opened when clicked.
- *Menus*. These offered users a structured way of choosing from a list of options or actions.
- *Pointer*. The visual feedback representing the input device (mouse, pen or trackpad), controlled by the user.

Although the building blocks stayed the same over the years, they have evolved into a more attractive and appealing design, more controls were created and more elements were added as building blocks, expanding the WIMP paradigm (such as toolbars and docks) [RSP11].

2.1.2.2 Limitations of Graphical User Interfaces

While GUIs brought decisive advantages to user interaction with the computer, there are some limitations to this interaction paradigm [Van97]:

- As an application becomes too complex, the interface tends to become harder to learn. This happens because of the profusion of controls and features;
- Expert users are often frustrated by too many layers of "point and click" and screen clutter due to too many controls, and prefer keyboard shortcuts;
- GUIs do not take advantage of speech, sound and touch. Although vision is our highest-bandwidth information channel, it is difficult for us to communicate in the physical world without these.
- GUIs are designed for a single desktop user who controls objects that have no autonomy and at most react to mouse manipulation.

2.1.3 Summary

The current interaction paradigm, GUI, has resisted for three decades. Despite its advantages and having a considerable impact in the HCI field and in our lives, it still has some limitations.

These limitations, in conjunction with new advances in technology, gave room for a new paradigm in Human-Computer Interaction: the Natural User Interfaces, described in section 2.4. It is not, however, expectable that GUIs will cease to exist, or even be relegated to a niche. They will still be used where they work best: in office environments, in conjunction with the mouse and the keyboard [WW11].

The next section presents an overview of Interaction Design, one of the disciplines of HCI.

2.2 Interaction Design

Interaction Design is the practice of designing interactive products, environments, systems and services. While it is also concerned with form, it is primarily concerned with designing the behavior. The focus of this practice is satisfying the needs and desires of the people who will interact with a product or service. [CRC07]

The principles of good interaction design found on everyday objects can and should be also applied to computers. In the book "Design of Everyday Things" [Nor02], Norman explains why

some products are pleasurable to use and others are a frustration, by criticizing bad designs and offering solutions for them. A set of principles and guidelines are identified, which can be applied to virtually any product used by humans. The same author criticized new gesture-based interfaces, labeled as "Natural User Interfaces"[2], as not natural [Nor10]. In the rush of experimenting new methods for the new technologies made available, well tested and understood standards were being ignored and violated. While new technologies require new methods of interaction, it is important to have these fundamental principles in mind, since they are independent of technology. Norman summarizes by recommending [NN10]:

> "We urgently need to return to our basics, developing usability guidelines for these systems that are based upon solid principles of interaction design, not on the whims of the company-interface guidelines and arbitrary ideas of developers."

In order to understand how systems can be designed for meeting the needs and expectations of users, it is important to understand how they think and reason, their conceptual model.

2.2.1 Conceptual Models

People have conceptual models of themselves, of others, the environment and the objects with which they interact. These are called mental models, and result from the tendency people have to form explanations for things. These models are formed through experience, training and instruction, and, as such, each individual has a different mental model, and different expectations as of how things work. Our mental models are often erroneously constructed, by means of fragmentary evidence, from our poor understanding of what is happening and from the causes, mechanisms and relationships we postulate, even where there are none [Nor02]. These mental models are what allows us to use a new, complex system, without knowing all the details of its operation and inner mechanism [CRC07].

Figure 2.2 depicts the different aspects of the conceptual models that act on a system [Nor02, CRC07]. The user interface (the designer model) must be designed to better match to what the users expects (the user mental model), and not to match how the system is implemented (the implementation model). The designer communicates with the user through the system image - the representation of the implementation model to the user. The designer must assure that the system evokes the appropriate system image to the user [Nor02]. In the figure, the system image should correspond to a circle, in order to be as close as possible to the user mental model.

The importance of designing a good conceptual model is summarized by Liddle [Win96]:

> "The most important thing to design properly is the user's conceptual model. Everything else should be subordinated to making that model clear, obvious, and substantial."

[2]Natural User Interfaces are the focus of section 2.4

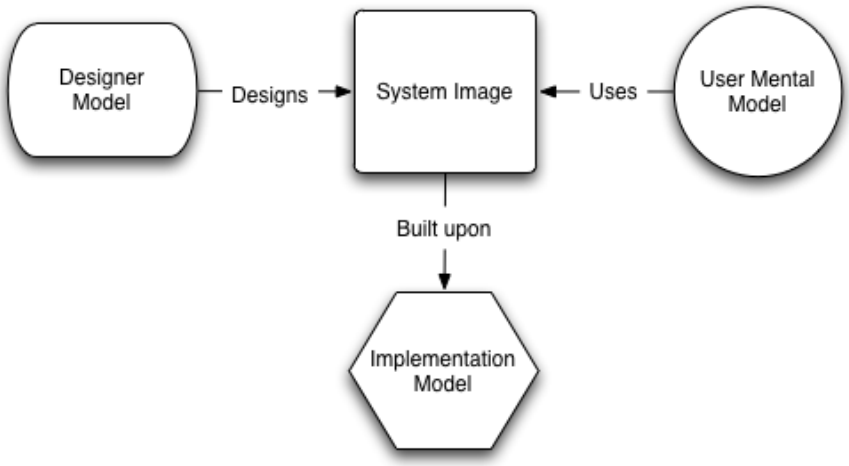

Figure 2.2: The conceptual models of a system

2.2.2 Principles and Guidelines for Interaction Design

There are several clues designers can put into a system, in order to make them aware of its underlying functionality and helping users form a good mental model of the system. There are several principles that are employed to support and influence user's mental models [DDW99, Nor02, Joh10, RSP11]:

- *Visibility*. Available actions should be made visible to the user, in order to allow the system to be easily discoverable.
- *Feedback*. A system should provide complete and continuous feedback about the results of the user's actions.
- *Consistency*. An interface should allow users to build on prior knowledge they have from experience interacting with the world. The interface should also be consistent in itself and with other systems the user might have used.
- *Simplicity*. Since mental models simplify reality, interface design should simplify actual computer functions. Include only what is needed.
- *Flexibility*. An interface should support alternate interaction techniques, allowing users to choose the method of interaction that is most appropriate to their situation.
- *Safety*. A user's action should cause the results the user expects, with actions that are revocable, so they do not cause irreversible consequences.
- *Affordance*. Affordance refers to the perceived and actual properties of the system. Good affordances provide clues on how an object can be used.

- *Constraints*. Constrain the way the system is used, in order to prevent the user from making errors.

Norman also offers the following guidelines, as a process for designers, in order to build good user interfaces, following the principles stated previously [Nor02]:

1. Use both knowledge in the world and knowledge in the head.
2. Simplify the structure of the tasks.
3. Make things visible.
4. Get the mappings right.
5. Exploit the power of constraints, both natural and artificial.
6. Design for error.
7. When all else fails, standardize.

These principles and guidelines should be taken into account when designing any kind of user interface. Norman states that the most important concepts to have into account are the following [Nor02]: conceptual models, feedback, constraints and affordances.

2.2.3 User Interface Metaphors

Metaphors are used in computer interfaces in order to aid users to understand a new target domain by allowing them to comprehend it in terms of a source domain they already comprehend [BGBG95]. This way, the designer model can be closer to the user model.

One example of metaphors is how the Xerox Star system was designed. Because it was targeted to workers not interested in computing, metaphors based on a physical office were created [RSP11]. As a physical desktop is used for organization and realization of office tasks, we can use the computer display as the desktop, where we can have documents and folders on the desktop. Opening a document will open a window, which can be moved around the desktop and overlapped with other windows, as a paper would.

Interface metaphors have proven to be highly successful, by providing users a familiar device and helping them understand and learn how to use a system. However, designers should have caution with their use. Sometimes designers make the mistake of creating an interface metaphor to look and behave exactly like their real entities. This could lead to designing a constraining interface, that conflicts with basic design principles and can even provide the user an erroneous mental model [RSP11]. An example is the Magic Cap interface that was based on literal metaphors for desktops, buildings, and villages. This introduced a level of clunky indirectness in achieving common user goals, since they are navigationally cumbersome [GN96].

2.2.4 User-Centered Design

User-Centered Design (UCD) is a philosophy that puts its emphasis on people, rather than technology [ND86]. Its goal is to design systems that enable users to achieve their goals and meet their needs in a particular context of use [Kai11].

ISO 9241-210 standard ("Human-centred Design for Interactive Systems") describes six principles that characterize UCD [Kai11]:

1. The design of interactive systems is based upon an explicit understanding of users, tasks, and environments.
2. Users are involved throughout the design and development.
3. Design is driven and refined by user-centered evaluation.
4. The process is iterative.
5. The design addresses the whole user experience.
6. The design team should be multidisciplinary.

Given the iterative nature of this process, and the need to evaluate various designs with the users, the development of prototypes became a common practice as part of User-Centered Design.

2.2.5 Summary

Following established interaction design principles is particularly important in designing novel user interfaces, such as Natural User Interfaces. These principles concern on making the system image closer to the user mental model, and making the experience of use pleasant and efficient.

The User-Centered Design philosophy is user-driven approach for designing a system, in which prototyping is an important activity.

Section 2.3, presents the importance of user interface prototyping, as well as an overview on the practice and on the existing tools that support it.

2.3 User Interface Prototyping

User Interface Prototyping is the process of iterative design and evaluation of the user interface of a product, before full development begins [KJ06, CH10].

A prototype can be described as a limited representation of a design, allowing users to interact with the representation of an envisioned product and to explore imagined uses. They are an useful aid for exchanging ideas and to communicate with stakeholders and team members [RSP11].

There are various ways of designing and presenting user interface prototypes. In terms of UI prototype design, it is important to distinguish between two kinds of prototypes, in terms of fidelity to the final system [RSP11]:

- **low-fidelity prototypes**, intended to be simple, cheap and quick to produce. They can be used early in the design cycle and are very useful for exploring ideas.
- **high-fidelity prototypes**, produce a product that looks more like the final product, using similar, or even the same materials. These prototypes should only be used for marketing purposes, or for testing specific technical issues, given that they are very expensive to create.

This dissertation focuses on low-fidelity prototypes. These prototypes have several advantages [CH10, KJ06]:

- they can be constructed at the early stages of the development cycle.
- they can be created quickly and at a low cost.
- users are more inclined to offer feedback, since they understand the interface presented is not final and easily changed.

Considering the cost of fixing design problems on the later stages of development [McC04, War09], low-fidelity prototypes are a valuable resource, capable of greatly improving user satisfaction, while lowering development cost [War09, RSI96, KJ06].

There are several techniques and tools for creating low-fidelity prototypes. For the purpose of this dissertation, these will be divided into two major groups:

- **Paper-based prototypes**, representing prototypes sketched or modeled on paper.
- **Software-based prototypes**, representing prototypes created using software tools, on a digital format.

2.3.1 Paper-based Prototypes

Despite the wide range of software tools that can be used for creating prototypes, two surveys done independently in 2008 [CH10, War09] agree that paper-based prototyping is the most used method of prototyping. Designers can sketch quickly their ideas, immediately translating their ideas into a visual medium and immediately manipulate it to respond to user feedback. Paper is also a very versatile medium, allowing to create simple sketches or mockups for complex interactions [War09]. Due to its versatility, paper prototyping was used to prototype multi-touch interactions [Van11, BPF09].

There are several representations of prototypes that can be done on paper. Some examples are [RSP11]:

1. *Storyboards*, a series of sketches describing how a user might progress through a task.
2. *Sketches*, representations of the proposed interface, in which the elements are drawn in a sketchy way, to illustrate the idea.

3. *Index Cards*, a series of small pieces of cardboard representing a screen or element of a task. Users can step through these, pretending to perform the task.

Paper-based prototypes can also be used for testing the interface with users. Sneyder summarizes this process in the book "Paper Prototyping" [Sny03]:

> "Paper prototyping is a variation of usability testing where representative users perform realistic tasks by interacting with a paper version of the interface that is manipulated by a person 'playing [the role of the] computer,' who doesn't explain how the interface is intended to work."

This technique is also known as *Wizard of Oz* prototyping, which is particularly useful for understanding user behavior and preferred modes of interaction, without being constrained by technological limitations [DML+05].

2.3.2 Software-based Prototypes

Software-based prototypes have several advantages over paper-based prototypes, such as offering designers multi-level undo, version control, virtual distribution and collaboration. The prototypes produced have a more consistent design and, for usability testing, subjects seem to prefer software-based prototypes over paper-based ones [STG03]. However, in terms of results, the quantity and quality of critical user statements and problems found is almost the same [STG03].

Prototypes can be done in several software packages not targeted to the creation of prototypes. It is common to create prototypes in graphics editing software, presentation software and on programming languages [CH10]. There also exist tools specifically targeted for low-fidelity prototype creation.

A comparative study of the functionality on several tools, selected because of their popularity and relevance of their features, was performed. Table 2.1 lists the tools considered, and Table 2.2 describes the functionalities analyzed. The comparison is presented in Table 2.3.

By analyzing Table 2.3, the basic operations the user expects, found in almost all prototyping tools, such as Locking, Grouping, Element Movement and Scaling, were compiled. The presence of a Grid is one important feature for guaranteeing a more consistent and easy placement of the elements. Novel features, such as Element Creation Gestures and Freehand Sketching, present some NUI elements, and are good candidates for helping solve the problems at hand. The other features were taken into consideration during the process of development of the solution.

One interesting tool that was omitted from this comparison was *Proxi-Sketch* [AGWF11]. This tool was built for a multi-touch tabletop and follows some NUI principles. However, it was built as a proof of concept for exploring *Medusa*, a proximity-aware tabletop, and not as a prototyping tool, being very limited in terms of functionality.

Table 2.1: List of the prototyping tools analyzed

Ref	Name	Input	Version
BM	Balsamiq Mockups	Mouse	2.1
MB	Mocking Bird	Mouse	Web
FB	Flair Builder	Mouse	3.0
ES	Expression Sketchflow	Mouse	2.0
AX	Axure RP Pro	Mouse	6.0
MT	Mockup Tiger	Mouse	Web
PE	Pencil	Mouse	1.3
MO	Mockups for Android	Touch	1.4
AP	Adobe Proto	Touch	1.0
MM	Mockups.me	Touch [1]	1.4
DE	DENIM	Pen	2.1

[1]The Desktop Trial version was used for the test, but we considered the Tablet version had the same features, plus the advertised "Sketch Mode" feature.

2.3.3 Summary

Prototyping is a valuable resource for User-Centered Design. Paper-based prototyping remains the most used method of prototyping, in prejudice of software-based ones. This is due to the versatile, natural and direct way of manipulating paper. Natural User Interfaces, the focus of the next section, can be a bridge between paper and software. Although there are already tools that are built for other input interfaces than the mouse, they were not built to fully take advantage of the NUI paradigm, and are prepared to run in a tablet, instead of a multi-touch tabletop. They do, however, implement some interesting features, such as Element Creation Gestures and Freehand Sketching.

The analysis made in this section, combined with some brainstorming, provided a starting point for specifying the solution proposed in chapter 3.

Table 2.2: Description of the functionality offered by prototyping tools

Functionality	**Description**
Element Area	Area with widgets, elements and/or tools.
Design Area	Area where the prototype is designed.
Preview Mode	Mode for previewing and/or presenting the prototype.
Element Text Search	Filter elements by text searching (not to be confused with Quick Add).
Element Categories	Elements/widgets are grouped by category.
Grouping	Grouping/Ungrouping of elements, to be manipulated as a single element.
Layering	Elements can be put on front/back in relation to other elements (z-index).
Object Snapping	When moving elements in the design area, guidelines are presented in order to position the element aligned relatively to other elements
Element Movement	Translation of elements in the design area.
Element Scaling	Scaling of elements, in the design area.
Element Locking	Locking/Unlocking of elements, disabling the manipulation of that element.
Group Alignment	Aligns or Distributes various elements in relation to each other.
Element Customization	Allows some degree of customization of each element (color, show scrollbars, etc.).
Quick Add	Quickly adds an element by inputing its name on the keyboard (similar to auto-complete on websites).
Flow Diagram	A diagram that shows (and allows editing of) the relations between each screen of the prototype.
Grid View	Shows a grid for better aligning/placing elements on the design view.
Freehand Sketching	Allows free sketching, for creating new elements.
Annotations	Allows viewers of the prototype to add their comments/annotations.
Images Insertion	Designers can add their own elements by importing an image from the filesystem.
Resource Navigator	Designers can select each resource in the designer view from a list.
Create Composition Element	Designers can create and reuse their own element by grouping various elements.
Animation Transitions	Designers can define user interaction via transitions between states, that can be triggered through events.
Element Cloning	Allows duplication of selected elements.
Element Creation Gestures	Symbolic gestures can be performed to quickly add elements on the screen.

Table 2.3: Functionality offered by prototyping tools

Functionality	BM	MB	FB	ES	AX	MT	PE	MO	AP	MM	DE
Element Area	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Design Area	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Preview Mode	✓	✓	✓	✓	✓	✓	-	-	✓	✓	✓
Element Text Search	-	✓	-	✓	✓	-	✓	-	-	-	-
Element Categories	✓	✓	✓	✓	-	✓	✓	-	✓	✓	-
Grouping	✓	✓	✓	✓	✓	✓	✓	-	✓	✓	-
Layering	✓	✓	✓	✓	✓	✓	✓	-	✓	✓	-
Element Relative Positioning	✓	✓	✓	✓	-	-	✓	-	✓	-	-
Element Movement	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-
Element Scaling	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-
Element Locking	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-
Group Alignment	✓	-	✓	✓	✓	✓	✓	-	-	✓	-
Element Customization	✓	-	✓	✓	✓	✓	✓	✓	✓	✓	✓
Quick Add	✓	-	-	-	-	✓	-	-	-	✓	-
Flow Diagram	-	-	✓	✓	-	-	-	-	-	-	✓
Grid View	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-
Freehand Sketching	-	-	-	✓	-	-	-	-	✓	-	✓
Annotations	✓	-	✓	✓	✓	-	-	-	-	✓	-
Images Insertion	-	-	-	✓	✓	✓	✓	-	-	✓	-
Resource Navigator	-	-	-	✓	-	-	-	-	-	✓	-
Create Composition Element	-	-	✓	✓	-	-	-	-	-	✓	-
Animation Transitions	-	-	✓	✓	✓	-	-	-	-	✓	-
Element Cloning	✓	✓	✓	✓	✓	✓	-	-	-	✓	-
Element Creation Gestures	-	-	-	-	-	-	-	-	✓	✓	✓

2.4 Natural User Interfaces

In the last two decades, new technologies, offering new ways for user interaction, have been developed. An example is the wide availability of multi-touch technologies, ubiquitous in consumer smartphones and laptops. Other developments include virtual, mixed and augmented reality, in-air gesturing, tangible interaction and voice interfaces [JGH+08, WW11, RSP11].

What these interaction styles have in common is that they build upon the user knowledge of the real world [JGH+08]. This has the promise of further lowering the barriers of computing, while increasing the power the user has over the system. They can potentially change the paradigm of user interaction, similar to the way GUI has evolved over CLI systems [WW11, Kra10]. However, these technologies alone do not guarantee a better user experience. It is the way the user interface and the user experience is designed, how they leverage these new technologies, that drives a new paradigm in user interaction [WW11]. The design goal should be to make these technologies better mirror human capabilities, apply them to given contexts and tasks, and fulfill user needs. In short, the goal is to make the user feel natural about using them [WW11]. This is the basis of Natural User Interfaces (NUI).

The term Natural User Interfaces has been applied in diverse contexts. In 1988, Buxton, one of the pioneers of NUIs and multi-touch technologies, stated that "user interfaces should be more natural" [Bux88], and that designers should explore creative alternatives to the GUI [Bux90]. More recently, the same author described Natural User Interfaces as those that "exploit skills that we have acquired through a lifetime of living in this world" [GB10].

Natural User Interfaces do not, however, have the goal to mimic the real world into its experience. Its main design goal is to provide a more natural way for users to interact with the product. This can be achieved by designing the product in such a way that users feel "at home" the moment they start using it. As such, the "natural" in Natural User Interfaces does not refer to the interface design of the product, it refers to the way a user behaves and feels during the process. This is summarized by Wigdor and Wixon as [WW11]:

> "A NUI is not a natural *user interface*, but rather an interface that makes your user act and feel like a natural. An easy way of remembering this is to change the way you say "natural user interface" - it's not a *natural* user interface, but rather a *natural user* interface."

2.4.1 NUI Elements and Design Principles

Natural User Interfaces are still in their infancy, without well-established principles and guidelines. The principles presented here should be taken as a design philosophy and design goal, rather than as a design recipe.

A UI can be defined as a NUI if it provides a User Experience (UX) that encompasses *all* of the following elements [WW11]:

- It is enjoyable;
- It leads to skilled practice;
- It is appropriate to context.

In order to design a user experience that accomplishes those elements, there are seven key principles to guide us [WW11, Mic09]:

2.4.1.1 Seamless

In a seamless experience, users are "cognitively and emotionally immersed so that they embrace these new experiences and rapidly progress to skilled practice" [WW11]. To achieve this, the experience must be capable of suspending the user's sense of disbelief, making them accept something as real or possible, even if it is not possible in the real world. The system must continuously respond to user input and provide appropriate feedback, by displaying information in expected ways. Special care should be taken in design and performance, since any disturbance on the system, if it is slow, or responds unexpectedly, will break the seamless experience.

2.4.1.2 Social

A NUI should be designed for multi-user input. It should also promote the interaction between individuals, away from the computer interface. That means that communication should happen more among individuals around the user interface than between an individual *and* the user interface.

2.4.1.3 Spatial

NUI environments encourage immersion, by making objects appear to have volume or basing their manipulation in real-world 3D behaviors. There are situations where 3D environments should not be considered, as they can be disorienting and complex. However, the z-axis should always be considered. For instance objects can "rise" when the user touches them. The environment can be extended past the screen boundaries, allowing users to relocate content by interacting with the environment, and can be organized in a 3D space. This can better leverage the user's sense of depth and spatial memory.

2.4.1.4 Simple

The design should start in simple fundamental interactions. After these are perfected through testing, designers should build on them, and extrapolate them into more complex functionality.

2.4.1.5 Super Real

Super realism acts as an intuitive extension of the real plane, pushing beyond the physically natural world, to provide experiences that do more than what is possible in the real world. An example

of this principle is the "pinch to zoom" gesture commonly found on multi-touch interfaces. In the real world, stretching a picture would break it, but in the virtual (super real) world, the images expands (scales) with our fingers, providing an extension that is intuitive and appreciated by the users.

NUIs should be designed such that they work as users expect them to work. And as we try something impossible, but plausible, users should discover that it works too.

2.4.1.6 Contextual Environments

A NUI does not aim to magically respond to any action we make, guessing the user intent. Instead, they are efficient in their context and environment, by eliciting appropriate actions from users and shaping them into skilled behavior in a smooth, efficient and enjoyable way.

2.4.1.7 Scaffolding

Scaffolding is "the creation of a design that promotes autonomous learning by employing actions that encourages users to develop their own cognitive, affective, and psychomotor skills" [WW11].

Scaffolding provides supportive structures and situations to encourage active exploration. Users should be presented only with the reasonable choices for that specific moments. This way they can be more efficient, by using simple processes and tasks.

2.4.2 The OCGM as a Post-WIMP Interaction Style

In the same way as the WIMP interaction style was useful for the GUI to establish foundational metaphors of interaction, a new style should be considered for NUI. There are several candidates for the new acronym [MH09], each having its own rationale. For the purposes of this dissertation, the OCGM acronym, developed by George and Blake [GB10], will be used to describe the foundational metaphors of NUIs.

OCGM stands for "Objects, Containers, Gestures and Manipulations". It should be pronounced "Occam", as in "Occam's Razor" [GB10]. This analogy with the "Occam's Razor" is appropriate, since OCGM is based upon the simplest human interactions and our earliest thought patterns [GB10]. In this interaction style:

- *Objects* are metaphors for units of content or data.
- *Containers* are metaphors for the relationships between content.
- *Gestures* are metaphors for discrete, indirect, intelligent interaction.
- *Manipulations* are metaphors for continuous, direct, environmental interaction.

Based on the works of Piaget concerning the intellectual development of children [Pia70], the cognitive skills required to understand and operate WIMP interfaces are initially developed during or after pre-school age, while cognitive skills required for OCGM interfaces are developed

significantly earlier, by nine months old. As such, interfaces using OCGM will have minimal cognitive load and use skill-based behaviors [GB10].

However, there is not a consensus on its use. For instance, Jetter et al. consider OCGM gestures as symbolic, remnants of the conversation metaphor, and as such, not natural [JGR10]. They represent a sign language, which users have to learn. But taking into account the definition of the NUI, and the scaffolding and super real principles, symbolic gestures have the potential to be truly useful, if the user is properly guided into learning how to use them.

2.4.3 Summary

There are several interaction forms that support a Natural User Interface design (it could be argued that all forms of interaction support the design philosophy of NUI, but some forms better leverage this design goal than others). This dissertation only focuses on multi-touch interaction, which will be the focus of section 2.5.

The OCGM interaction style has the potential to be applied for designing the solution to be generic enough for freeing designer imagination. Despite only being a recent proposal, it provides an useful set of metaphors.

2.5 Multi-touch Interaction

A multi-touch screen or display is a device that allows direct touch interaction with the elements displayed, by recognizing more than one point of contact. This definition should not be confused with multi-touch trackpads, which provide indirect touch interaction with the system, or single touch devices, which only recognize one finger at a time [Bux11].

Multi-touch interaction has been researched since the 1980s, while touch displays have been studied since late 1960s. However, it took twenty-five years for this technology to be mass-marketed, with the release in 2007 of the Apple iPhone [Bux11, Bux10].

There are several technical aspects of multi-touch interaction that can leverage a Natural User Interface design, in contrast with the indirect "point-and-click" interaction of the computer mouse. These are [KAD09, FWSB07, HMD08, WW11]:

- *Direct manipulation*. In contrast with the mouse pointer, touch allows for a more natural way of interaction, by directly manipulating the graphical elements on the display.
- *Multi-user collaboration and awareness*. By allowing multiple points of input, large multi-touch displays facilitate co-located multi-user collaboration. The users are also aware of the movements and activities of other users, supporting fluid interaction and coordination.
- *Higher input bandwidth*. Multiple points of contact allow an increased input bandwidth.

There are also several limitations on multi-touch displays, which deserve the designer's attention when designing for these devices. These are [SG03, HMD08, GJS10]:

- *Unintentional interaction*. The user may unintentionally trigger events in the system, by accidentally touching the display.
- *Lack of precision*. Fingers are more error-prone for precise operations.
- *Occlusion*. Fingers, hands or even the arms can occlude the objects on the screen.
- *Screen orientation*. When various users are present, it is not easy to calculate the best orientation to show the information.
- *Lack of touch feedback*. Touch interaction heavily relies on visual feedback. One example of this is typing: in ordinary cellphones it was possible to type by feel, without looking, which is practically impossible on a touch-based screen.

It is important to have these advantages and limitations in mind when designing for these devices.

2.5.1 Multi-touch Hardware

There are various technologies for producing a multi-touch display. It is not the goal of this dissertation to detail each one, but to provide an overview of each, with their limitations and possibilities. These technologies can be grouped into three types [SBD+08]:

- *Resistance-Based Touch Surfaces (Resistive)*. This technology consists on two conductive layers separated by an insulating layer. A controller alternates between the layers, driving one with an electrical current and measuring the current of the other. Thus, when the user touches the display, the conductive layers are connected, establishing an electric current that is measured horizontally and vertically, to calculate the point of contact (Figure 2.3 (a)).
- *Capacitance-Based Touch Surfaces (Capacitive)*. This technique is based on the change of electrical properties in the surface. When the finger or other conductive object touches the screen, a small transport of charge is passed from the surface to the touching object. This is then measured to provide an accurate localization of the point of contact (Figure 2.3 (b)). This technique can be divided into two classes: Surface Capacitance and Projected Capacitance.
- *Optical Based Touch Surfaces (Optical)*. These technologies share the same approach of processing and filtering captured images. They are very popular due to their scalability, low cost and ease of setup [NUI09b]. There are various optical approaches: Frustrated Total Internal Reflection (FTIR), Diffused Illumination (DI), Laser Light Plane (LLP), Diffused Surface Illumination (DSI), LED Light Plane (LED-LP). These are documented, with their advantages and disadvantages in [NUI09b]. Figure 2.4 illustrates the schematics for a FTIR setup.

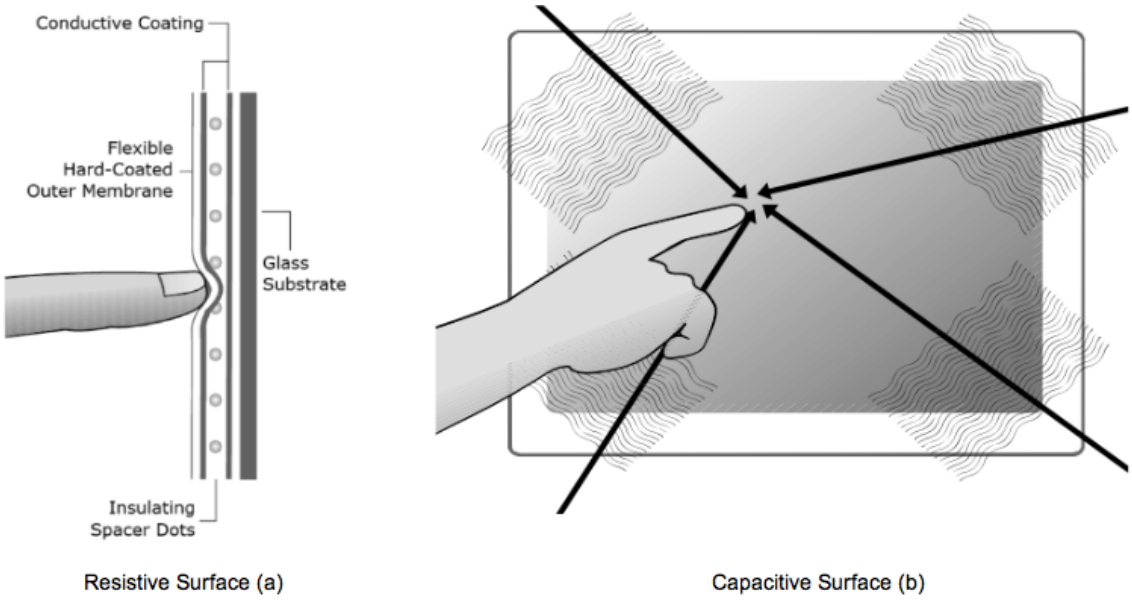

Figure 2.3: Schematic construction of a touch screen based on resistive technology (a) and representation of how the position of a touch is measured in a capacitive surface(b): Electrodes around the edges distribute voltage across the conductive layer creating an electric field. Touching the panel results in current drawn from each corner which is measured to define the position. [SBD+08]

2.5.1.1 Multi-touch Hardware Communication Protocols

Windows 7 Touch [Mic12] and MPX [Hut10] offer hardware communication protocols supporting multi-touch events, for hardware with compatible drivers. These protocols are, however, specific to an Operating System.

In an attempt to provide a general communication interface between different multi-touch hardware and operating system, Kaltenbrunner et al. proposed the TUIO Protocol [KBBC05]. This protocol allows tracking of finger and hand gestures, as well as tangible objects on the surface. It was widely adopted by the community and can be considered a community standard [Kal10].

2.5.2 Multi-touch Design Practices

There are several design practices and recommendations to follow when designing for multi-touch displays. Since this is an area in constant evolution, little is standardized and these principles do not intend to pose as a definite solution for multi-touch design. However, they were the result of research and experiments and, as such, are a good starting point.

2.5.2.1 Do not design for mouse input

The WIMP interaction style is a mature style and appropriate to the point-and-click interaction of the mouse. This happened because the WIMP was designed from scratch for mouse input.

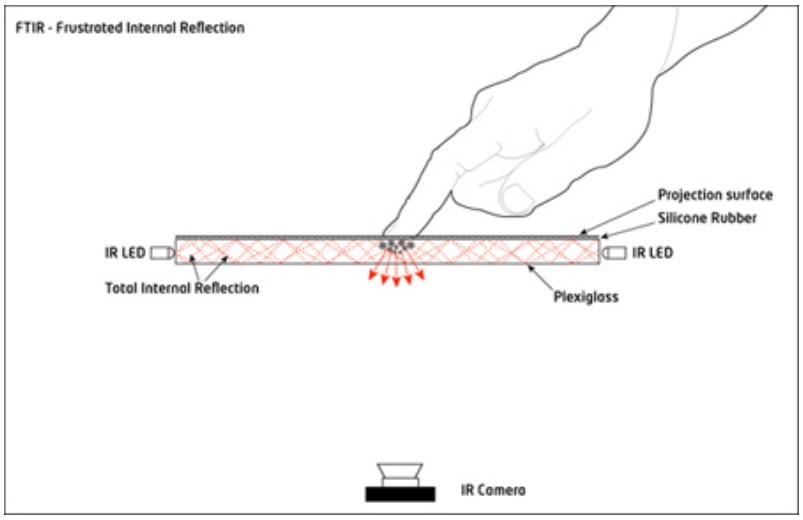

Figure 2.4: FTIR schematic diagram depicting the bare minimum of parts needed for a FTIR setup[NUI09b]

Although it is possible to emulate the mouse input in a multi-touch device, this approach should be discouraged. Mouse and multi-touch input are very different and, as such, for the user interface to be successful, a UI for a multi-touch device must be designed from the ground up [WW11]. By emulating the mouse, the information the computer has about the interaction is reduced to the x and y coordinates of the pointer and the button's state.

"Everything is best for something and worst for something else". This saying by Buxton, one of the multi-touch and NUI pioneers, illustrates why designing for the WIMP interaction style and translating that to a multi-touch input is a bad idea [Bux11]. The resulting interface would be limited in possibilities and suboptimal for that device. An all new set of primitives and controls must be defined for the interaction language [WW11].

2.5.2.2 Precise Selection

The precise selection of an element is one of the drawbacks of touch interfaces, relatively to the same process with the mouse. A mouse can provide pixel-accurate selection, by having a visual pointer to the pixel the user wants to select. The area a finger covers in a selection, and the lack of visual feedback showing the area that will be selected are problems in direct-touch interaction. Fortunately, there are some solutions proven to work [WW11]:

1. *Larger Objects*. For large touchscreens, designing the target size for a minimum of 1.6 cms is a good practice [WW11].

2. *Iceberg Targets*. This technique makes the area around an on-screen target to be selectable, not only the displayable object itself. A variation of this technique is to select the closest object to the selected pixel.

3. *Lifting*. The selection event can be changed to when the user lifts his finger. When the finger lands, visual feedback could be given to indicate what element will be selected, while lifting would select it.

4. *Escaping*. This technique, illustrated in Figure 2.5, allows precise selection of spatially close elements.

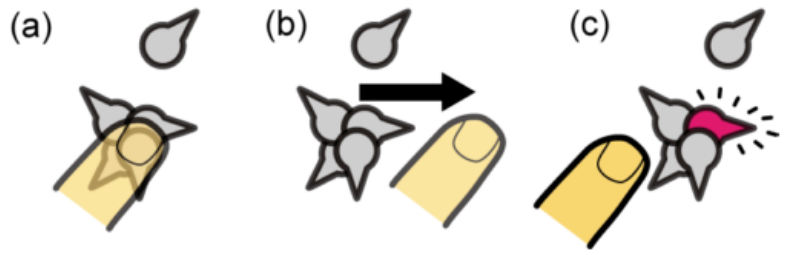

Figure 2.5: The Escape target selection technique. (a) The user presses her thumb near the desired target. (b) The user gestures in the direction indicated by the target. (c) The target is selected, despite several nearby distracters. [YPBN08]

2.5.2.3 Feedback is Essential

The principle of *feedback* - continuously showing the user the results of his actions - is critical for making any system usable [Nor02]. One of the main problems associated to multi-touch interfaces is the lack of feedback given by applications [Nor10, NN10]. This might come as a surprise, since, compared with mouse interfaces, the act of touching a surface is already giving us some feedback (both visual and tactile). However, the WIMP interaction style was designed with feedback in mind. One example is the visual representation of a pointer. If the system becomes non-responsive, the user can shake the mouse and see if the pointer moves on the screen. This is a solution provided by the Operating System. However, most touch-based system do not have a similar system: if the screen does not respond to their input, they will press it harder, wondering if the system has failed or if their input is at fault [WW11].

The freedom allowed by Natural User Interfaces and the lack of established guidelines, makes the responsibility of the designer to provide proper feedback for its applications. There are some guidelines to address the following sources of errors [WW11]:

- *Activation Event*. User feel the physical click of the mouse button when they depress it. On different touchscreens, some pressure may be needed to trigger the activation event, or it can even be triggered before the touch happens. For this reason, a correct feedback should indicate the activation event, to help the user to be more accurate.

- *Fat Fingers*. When the user misses a target, the feedback should show that the failure was due to a miss and, ideally, show the user how to avoid missing the target in the future.

- *Activation*. The system must provide immediate feedback as to whether the touch is on an active element or one that ignores the input.
- *Nonresponsive content*. Elements that do not expect user input must have proper affordances indicating that.
- *Accidental Activation*. When a user accidentally touches the screen, sudden state changes in the application will confuse the user, because most of the times they will not notice their input.
- *Multiple Capture States*. If the user tries to use a control intended for one finger with more fingers, there is no guarantee of what finger will be captured for mapping to that control. This is known as an *overcaptured* state, and the system should report this state to the user, so it can help him understand this.
- *Physical Manipulation Constraints*. When an element is moved or scaled past its constraints, appropriate feedback should be given to indicate this.
- *Interaction at a Distance*. Use of controls can extend beyond the bounds of those controls. An example is the WIMP scrollbar, which allows the pointer to move horizontally beyond its bounds after selections, since it only captures vertical movement. Special care should be taken for multiple fingers/users in this context.

A more thorough explanation of the sources of errors enumerated above and detailed solutions are referenced in chapter 14 of the book "Brave NUI World" [WW11].

2.5.3 Gesture Design

Multi-touch interfaces need to break away from the point-and-click paradigm. Designers should think of gestures and manipulations as the primitives of interaction for these devices.

2.5.3.1 Gesture Representation

It is useful to define a language for gesture design, in order to document and communicate gestures. This language would ideally be sufficiently generic to allow the specification of every kind of gestures that can be accomplished in a multi-touch table, while being compact and computable.

Several languages that try to accomplish these requirements have been researched and proposed. However, none of these can be considered as a standard or proven solution.

- Lao et al. [LHZ+09] propose a design model for touch interaction based on three structured levels and their relationships, in order to allow re-use of touch for different applications, platforms and use contexts. They also present a unified definition and description of all possible touch gestures and the logic flow of a gesture recognition algorithm based on their model.

- *GeForMT (Gesture Formalization for Multi-Touch).* This is a proposed formalization for multi-touch gestures based on a rationale rooted in semiotics [KWK+10].
- *Proton.* This is a framework that allows the declarative specification of a gesture as a regular expression over a stream of touch events [KHD12]. It also includes a Gesture Tablature Editor (Figure 2.6), providing visual design of the gestures as regular expressions.
- *GDL(Gesture Definition Language)*, proposed by Khandkar and Maurer [KM10], is a domain-specific language to define gestures for multi-user, multi-touch scenarios.
- *GDML (Gesture Definition Markup Language).* Describes gestures and their characteristics using an XML syntax [NUI09a].
- Gorg et al. present an approach for abstract gesture representation and recognition by means of mathematical rule calculus [GCG10].

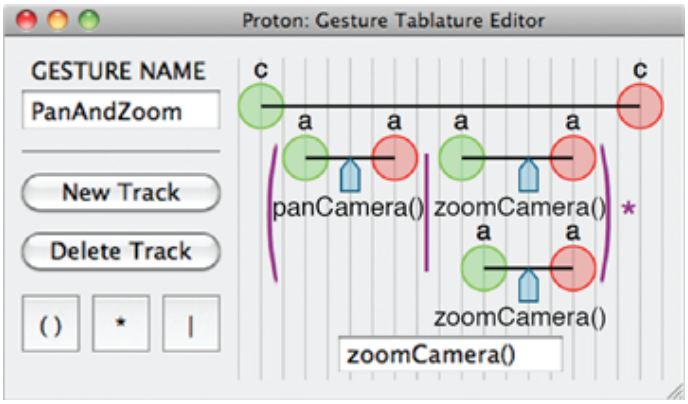

Figure 2.6: Proton's Gesture Tablature Editor [KHD12]

2.5.3.2 Gesture Recognition Methods

The approaches enumerated in section 2.5.3.1 refer to gesture recognition in terms of their representation, without considering technical details such as normalization and probabilistic analysis.

There are various techniques for gesture recognition, based on different approaches [Nar08]: *Hidden Markov Models* [LK99, Yan94, OSK02, WKZ08], *Artificial Neural Networks* [JMRW01], *Statistical Classifiers* [Cho06, Rub91], *Stochastic Methods* [JDM00] and *Signal Processing* [ZL01].

However, these techniques are very complex to implement [WWL07]. A simpler algorithm for one stroke gesture recognition, $1 recognizer, was proposed as an "easy, cheap and usable almost anywhere" algorithm, offering great accuracy and versatility, despite its simplicity [WWL07]. Another algorithm, Protractor, is similar to $1, but provided faster and more accurate results [Li10]. A variation of the $1 recognizer for multi-stroke recognition, the $N recognizer, was also proposed [AW10].

The simpler algorithms, such as the $1 recognizer, performed similarly or even better than complex algorithms, offering an accuracy of 99+% with only a few loaded templates [WWL07]. This makes them a good candidate for this dissertation.

2.5.3.3 Gesture Design Guidelines

Studies made for discovering gestures that were natural and intuitive, by allowing users to freely manipulate a multi-touch display, reached an important conclusion: there are no gestures that could be considered natural or intuitive for a majority of users [WMW09, HC11]. The gestures people use are influenced by their interaction and social context, and as such we should assume that every gesture will be learned and not guessed by the user [WW11, WMW09].

Guidelines exist to help designers in designing more easily learned and efficient gestures:

- *Gesture reuse*. There are various collections of gestures, the result of experimentation and research, that designers can use [EWH07, WMW09, VWW10, Mic09]. It is expectable that, in the future, gestures will be standardized [LHZ+09, Nor10, WG10]. Gesture designers should start by studying successful gestures in similar contexts, in order to apply them to their own context.

- *Registration, Relaxation and Reuse*. Wu et al. [WRFB06] propose a set of gesture design principles. The registration phase is the beginning phase of every gesture operation, setting the context for subsequent interactions. A distinctive hand posture is important in this phase. After this phase, the shape and dynamics of the gesture can be relaxed, in order to unburden the hand posture, allowing the gesture to be performed with minimal constraints after it is registered. The reuse of primitives is encouraged, enabling to build larger sets of gestures without requiring additional primitives to be defined.

- *Identity, Negation, Reciprocity, Commutative*. Wigdor and Wixon [WW11] applied genetic epistemology of cognition to gestural systems, assuming that an easy-to-learn system operates logically in a way that is analogous to the human reasoning. Gestures could be built on this principles of Piaget's concept of Identity, Negation, Reciprocity and Commutative.

- *Avoiding false gesture recognition*. The gesture language must be carefully balanced in order to minimize false-positives errors, when the software triggers recognition of a gesture where none was intended and false-negatives, when the user intends to perform a gesture but it is not recognized [WW11].

- *Self-revealing gestures*. Since gestures are not guessable, proper affordances should be given in order to guide the user into learning how to use the system, and to make the possible actions visible. Examples of these affordances are the techniques of "just-in-time chrome" [WW11], multi-touch marking menus [LGF10], based on pen-based marking menus [Kur93] and ShadowGuides, a gesture learning tool[Wig10].

2.5.4 Multi-touch Frameworks

There are various frameworks intended to help programmers build multi-touch applications. We analyzed some of the most popular frameworks, that are freely available. These are listed in Table 2.4. Table 2.5 presents our overview, which was based on the taxonomy done by Kammer et al. [KKFW10], but containing criteria that is useful for the purpose of this dissertation, and now updated with more recent frameworks, most notably Kivy, which is a replacement for the PyMT framework.

Table 2.4: List of the multi-touch frameworks analyzed (as of June 2012)

Name	**Version**	**Last Updated**	**URL**
Kivy	1.3.0	Jun 2012	http://www.kivy.org/
MT4j	0.98	Mar 2011	http://www.mt4j.org/
WPF	4	Apr 2010	http://windowsclient.net/
SparshUI	1.0	Oct 2008	http://code.google.com/p/sparsh-ui/
Breeze	1.0.5	Mar 2010	http://code.google.com/p/breezemultitouch/
MIRIA SDK	1.061 beta	Feb 2011	http://miria.codeplex.com/
Surface SDK	2.0	Feb 2012	http://msdn.com/windows/surface

Table 2.5: Comparison of Multi-touch Frameworks

Functionality	Kivy	MT4j	WPF + XNA	SparshUI + Qt	Breeze + WPF + XNA	MIRIA SDK + Silverlight	Surface SDK
Cross-Platform	✓	✓	-	✓	-	✓	-
TUIO	✓	✓	Indirect	✓	Indirect	✓	Indirect
WM_Touch	✓	✓	✓	-	✓	✓	✓
Language	Python	Java	C#	C++	C#	.NET	C#
Open-Source	✓	✓	-	✓	✓	✓	-
2D	✓	✓	✓	✓	✓	✓	✓
3D	Indirect	✓	✓	Indirect	✓	✓	✓
Touch Params	✓	✓	✓	-	✓	✓	✓
Gesture Params	-	✓	✓	-	✓	✓	✓

Kivy and MT4j provide support for a wide range of platforms and have the added bonus of being open-source. The main advantages of MT4j over Kivy would be the gesture parameters and the direct support of 3D rendering. However, these are not limiting for the problem at hand, and the fact that Kivy has an active community and a large user base, which makes this option more attractive than MT4j.

2.5.5 Prototyping Multi-touch Interactions

In order to minimize the cost of experimenting multi-touch interactions, low-fidelity prototyping can be a valuable tool for designing and evaluating these interfaces [DDV+10]. Traditionally,

paper prototypes are the preferred method for low-fidelity prototyping of multi-touch interfaces, due to its versatility and the lack of tool support for prototyping of these interfaces [HKSBM11, Van11, War09].

In an attempt to address this lack of tool support, Hosseini-Khayat et al. extended *ActiveStory Enhanced*, a tool for low-fidelity prototyping and usability testing, to provide support for touch-based interaction, by providing gesture definition for prototype elements [HKSBM11]. Bolchini et al. propose a technique, *"paper in screen"* prototyping [BPF09], which does not have any interaction, relying on the thinking out loud usability testing method. Konig et al. propose *Squidy* [KRR09, KRR10], an interaction library for rapid prototyping of multi-modal interfaces, using a visual design environment that combines the concepts of semantic zooming with visual dataflow programming.

2.5.6 Summary

In the section, a series of platforms and frameworks that provide/enable multi-touch functionality were presented. Their capabilities and limitations call for specific guidelines and practices for interaction design. Some approaches to resolve identified problems were detailed. Gestures provide a primitive of interaction and, as such, guidelines on gesture design and algorithms on gesture recognition were presented and discussed.

An overview of the multi-touch frameworks was made, since using one will accelerate the development of the solution proposed in chapter 3.

The last section discusses prototyping for a multi-touch environment. The lack of tool support is evident, validating the problem defined in the introduction.

Chapter 3

Solution Specification - Natural Protopying

This chapter presents an overview of the solution proposed by this dissertation to the problem identified in section 1.2, supported on the research on the state of the art done in chapter 2. The solution is named Natural Prototyping, a reference to the ideas proposed for Natural User Interfaces, in an attempt to provide a more natural, fun and effective way of designing prototypes. A brief overview of the solution is presented in Section 3.1, as well as a more detailed, but still informal overview of each of its main components: the Editor Tool and the Viewer Tool.

The approach followed for the development of the solution is described in Section 3.2, detailing the methodologies and the work plan for this dissertation.

Finally, a formal specification is presented in the form of user stories, detailed on Section 3.3. In an attempt to illustrate and better translate key ideas from these user stories, Section 3.4 presents low-fidelity prototypes, in the form of mockups.

3.1 Overview of the solution

The solution proposed addresses the identified needs of a better software tool for low-fidelity prototyping, based on Natural User Interfaces principles, designed for a multi-touch tabletop. This solution also addresses the need for a tool for creating prototypes that allow multi-touch interaction.

The rationale used for considering that a NUI can be better than a traditional WIMP GUI for prototyping is based on studies [CH10, War09] indicating that paper-based tools are used more frequently than software-based tools, for interface prototyping. As such, designing such a tool for multi-touch displays, with their touch and pen input capabilities, seems a logical step in providing a more natural, paper-like experience, while still having the efficiency and capabilities of software-based solutions.

The solution here described consists of two different tools: the *Editor Tool* and the *Viewer Tool*. The *Editor Tool*, which is targeted for designers and stakeholders, is responsible for providing a NUI approach for creating prototypes. The prototype produced in the Editor is then stored in a file, in order to be easily distributed and used. The Viewer Tool, which is targeted for end-users, test subjects and stakeholders, is responsible for immersing them into the prototype created in the Editor, while providing a way for collecting user feedback, which is very valuable for developing future iterations of the prototype, as specified in UCD.

3.1.1 Functional Prototype

The Editor and Viewer tools are specified in detail in Sections 3.1.2 and 3.1.3, and a functional prototype of a subset of the features was created. This functional prototype was implemented horizontally - focused on delivering more functionality with less detail - in order to experiment the maximum number of functionalities. By less detail we mean functionality implemented iteratively and adaptable to change, with room for further development and improvement.

The user experience had the goal of being pleasant, enjoyable and feel natural, as expected by NUI principles. The functional prototype has solid foundations for further development, instead of being a throwaway prototype.

3.1.2 Editor Tool

The Editor Tool is the application in which interaction designers create and manipulate the elements that compose a prototype (Figure 3.1). The goal of this application is to provide a more natural way to create and edit prototypes. In an effort to provide a rational feature list, an analysis of relevant software prototyping tools was presented in section 2.3.2. Through analysis of that list and brainstorming sessions, a new list of features was specified for the Editor application. The User-Centered Design process, followed for the implementation of this solution (as described in Section 3.2), advocates design driven by user evaluation. As such, the features presented here should be considered as a starting point.

- *Element Library*. The user has access to libraries of elements, either pre-defined libraries or user-defined.
- *Direct Manipulation of Elements*. Elements should be translated, scaled and rotated by means of direct manipulation.
- *Element Creation by Sketching*. The user can sketch, by finger or pen input, new elements to be added to the library.
- *Element Creation by Composition*. The user can create new elements as a composition of more primitive elements.
- *Grid View*. A grid should be presented, in order to allow proper precision on element placement and manipulation.

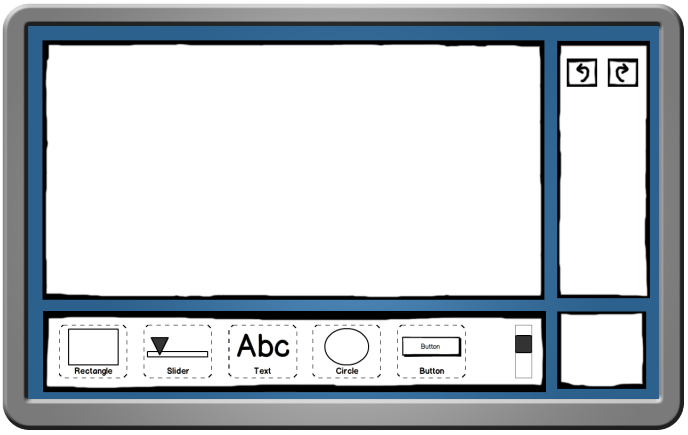

Figure 3.1: Editor Tool Overview - Mockup

- *Element Input by Symbolic Gestures*. Symbolic gestures can serve as shortcuts for element input. These gestures would ideally be customized by the user.
- *Element Snapping*. When manipulating objects, visual guidelines should be given in order to "snap" that element (relative alignment) to existing elements.
- *Manipulation Creation*. State-based transitions should be defined, in order to program the prototype to respond to user input.
- *Gesture Creation*. Users should be able to define their own gestures, for element input and for designing manipulations for user interaction in their prototype.
- *Element Inspection*. The user should be able to easily manipulate element attributes.

Besides these features, the editor should have common functionality found on other software tools: Undo/Redo, Group/Ungroup, Lock/Unlock, Alignment, Distribution, Layering and Cloning.

In order to understand how the application could be structured, the following areas are proposed:

- *Designer Area*. The area where the prototype is designed.
- *Element Library*. Organized library of elements, to be inserted in the Designer Area. For example, we could browse the library for a Text Field and place it on the Designer Area.
- *Toolbox*. An area for selecting different tools, such as Undo, Redo, Lock, Unlock and Freehand Sketching.
- *Screens Area*. An area for managing the screens that compose a project of interaction design.

The Editor is targeted for a multi-touch tabletop, and should be thought for multiple users working collaboratively, in the same physical space.

3.1.3 Viewer Tool

The Viewer application is where users and other stakeholders visualize and interact with the prototype created in the Editor. It could be considered as a prototype "player".

Besides the ability of showing the prototypes created, it should be capable of providing the interaction designed in the editor and the ability for annotations, so users can give impromptu feedback.

The Viewer is targeted for a wide variety of platforms, representing the platforms of the end-user. It should work on multi-touch tabletops, tablets, smartphones and desktops.

3.2 Methodology

For the implementation of the solution described in Section 3.1, a User-Centered Design approach is advocated, as described in section 2.2.4. In order to rapidly implement several iterations of the solution, a milestone-based approach was conceived for development of the solution, based on agile methodologies [BBvB+01, Hig09]. Each milestone had a development cycle with four phases, as described in Figure 3.2.

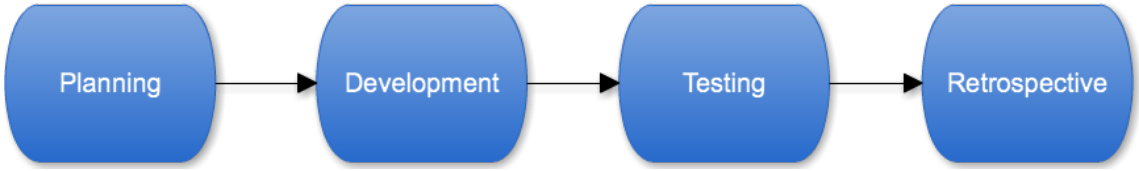

Figure 3.2: The cycle of development of a milestone

In the planning phase, user story elicitation and prototyping depicting those user stories was done, followed by planning of the development phase. The development phase was divided into major tasks or iterations. After development, a testing phase was completed, where acceptance tests were made to validate milestone results. The retrospect phase gathered insights from the milestone, for planning the next milestone.

The work done on this dissertation can be grouped into five major phases:

- State of the Art Review, which produced the Chapter 2 of this work, and set the foundations and principles in which this dissertation was developed;
- Architecture Specification, in which the architecture of the Editor Tool and the Viewer Tool was specified. The architecture is presented in Chapter 4;
- Milestone I, focused on implementing a base architecture for the Editor Tool, while delivering a functional proof of concept. Designers should be able to select from a small set of elements from the Element Library and place them in the Designer Area;

- Milestone II, focused on giving the users of the Editor Tool a more solid and complete experience, while providing additional features, such as manipulation/gesture attribution, multiple screen editing, project persistence and object scaling, layering and removal. Development on the Viewer Tool was started, accurately depicting the designs done with the Editor Tool;

- User Testing, in which the results of Milestone I and II were validated with users, as described in Chapter 6;

3.3 User Stories

This section presents an overview of the user stories specified for the solution proposed. It is important to notice that, given the scope of this dissertation, only a limited set of user stories were implemented in the functional prototype.

The user stories presented on this section focus on specifying a complete system, which would be implemented as a fully-featured solution for prototyping, given the appropriate resources. They are divided into six epics[1] for the Editor Tool: (Element Library, Designer Area, Toolbox, Interaction Design, Project Management and User Control) and one epic for the Viewer Tool. The user stories are listed in Table 3.1 and are described in more detail in Appendix A.

3.4 Mockups

The mockups herein presented were created with Balsamiq Mockups[2] and Adobe Photoshop[3], with the intent of illustrating key ideas of selected user stories[4].

Most of the mockups designed during the dissertation assumed a static Library, Toolbox and Designer Area. Two alternative scenarios were considered: a single-user scenario, where a static library and toolbox would optimize single-user interactions (by not filling the Designer Area), and a multi-user scenario, where each user has a Control Pad, described in Section 3.4.9, acting as a movable structure for the Library and Toolbox.

The overview presented in Figure 3.3 assumes a single-user static environment, which will be used for explanatory purposes, since the functionality and interaction is essentially the same. Section 3.4.9 simulates the collaborative environment with the Control Pad.

There are some interaction metaphors and devices used throughout the Editor, in order to provide users a basis to help them quickly become proficient with the system model (See Section 2.2.1 and Section 2.2.3). These are the Designer Area, Toolbox, Library, List, Marking Menu, Virtual Keyboard, Control Pad and Undo Confirmation Dialog.

[1] An *epic* is a large user story, that is divided into smaller user stories [Coh04].
[2] Balsamiq Mockups 2.1 - http://www.balsamiq.com/products/mockups
[3] Adobe Photoshop CS6 Beta - http://www.adobe.com/products/photoshop.html
[4] Illustrations provided by GestureWorks® (www.gestureworks.com)

Table 3.1: List of User Stories

Epic	Ref.	Name
Element Library	US-001	Element Library Organization
	US-002	Element Library Browsing
	US-003	Composite Element Creation
Designer Area	US-004	Freehand Sketching
	US-005	Element Placement
	US-006	Element Instance Direct Manipulation
	US-007	Element Instance Properties
	US-008	Element Instance Layering
	US-009	Element Instance Removal
	US-010	Group Selection
	US-011	Grid Navigation
	US-012	Element Cloning
Toolbox	US-013	Undo/Redo
	US-014	Alignment/Distribution
	US-015	Lock/Unlock
Interaction Design	US-016	Gesture Definition Library
	US-017	Action Library
	US-018	Attribute Gesture
	US-019	Manipulation Library
	US-020	Attribute Manipulation
Project Management	US-021	Create/Remove Screen
	US-022	Select Active Screen
User Control	US-023	Create New Control Pad
	US-024	Move Control Pad
	US-025	Dismiss Control Pad
Viewer Tool	US-026	Start Prototype
	US-027	Annotate Prototype

3.4.1 Designer Area

- **Related To:** EPIC-002
- **Description:** A WYSIWYG area that represents a screen of the prototype. This is the area where elements are placed and manipulated by the Designers. See Figure 3.3 - 1.

3.4.2 Toolbox

- **Related To:** EPIC-003, US-004
- **Description:** A container of several buttons, providing the Designer with visible commands so he can use them to manipulate the prototype. See Figure 3.3 - 3.

3.4.3 Library

- **Used in:** Element Library, Gesture Library, Action Library, Manipulation Library

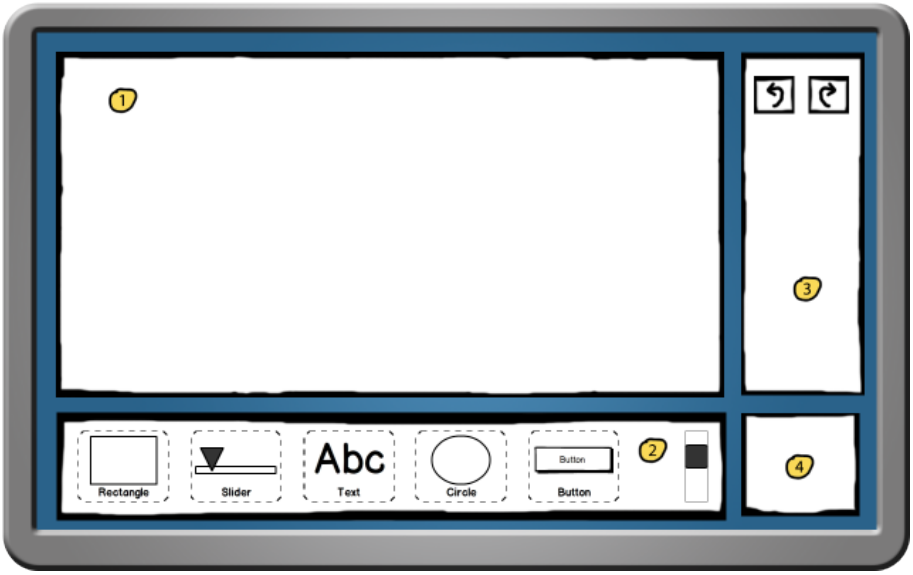

Figure 3.3: Overview of the Editor Tool: 1) Designer Area; 2) Element Library; 3) Toolbox; 4) Screen Overview.

- **Related To:** EPIC-001, US-005, EPIC-004
- **Description:** The goal of the library is to provide quick access to a large collection of items. The items are organized by groups. The user can customize the ordering of the groups and of the items inside the groups. See Figure 3.3 - 2.

Figure 3.4 shows an example of changing the ordering of an item, swapping items inside the same group. Figure 3.5 shows an example of changing of ordering an item, swapping items from different groups. Figure 3.6 shows an example of changing the ordering of two groups, by swapping them.

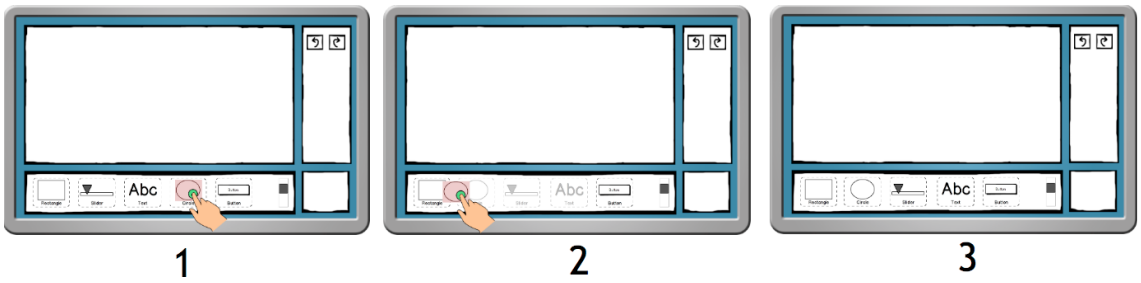

Figure 3.4: Item Ordering - Same Group Mockup

3.4.4 List

- **Related To:** US-007, US-017, US-019

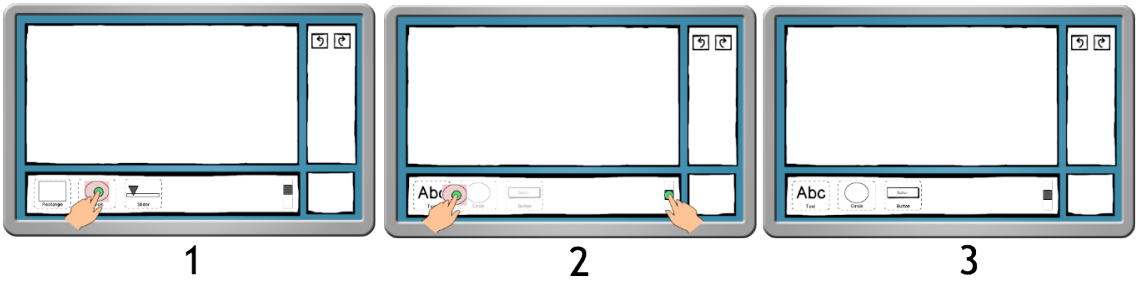

Figure 3.5: Item Ordering - Different Groups Mockup

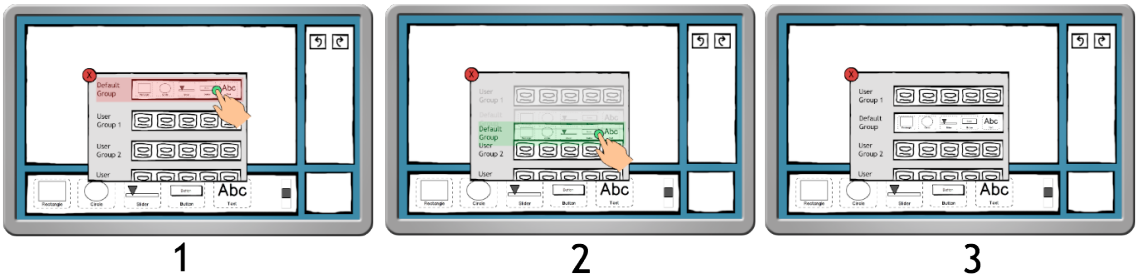

Figure 3.6: Group Ordering Mockup

- **Description:** A list provides a sequence or set of items, that are attached to an object.

  Figure 3.7 shows the Element Inspector, a List which has Element Properties as items.

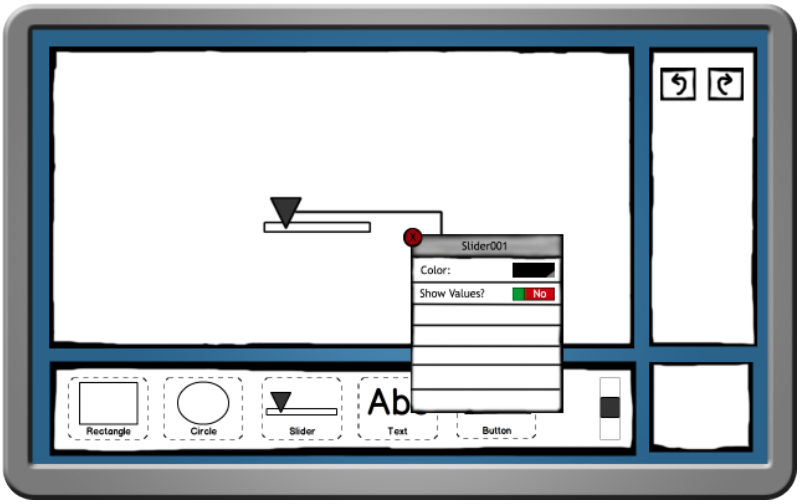

Figure 3.7: Inspector List Mockup

3.4.5 Marking Menu

- **Related To:** US-001, US-007, US-009

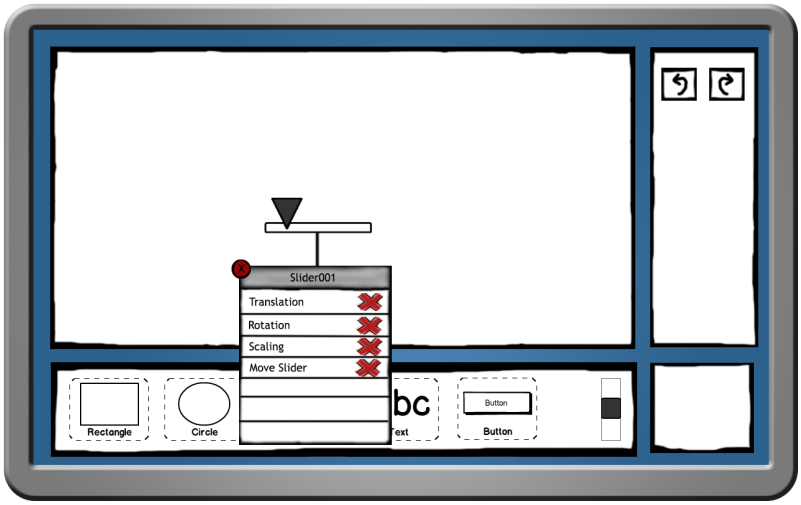

Figure 3.8: Manipulation List Mockup

- **Description:** A menu metaphor, described in Section 2.5.3.3 (self-revealing gestures), which provides contextual options for a selected object.

The Marking Menu is called by holding five fingers on top of an element. The element is selected based on the center of the circle defined by the 5 fingers. There are two ways to select a marking menu option:

- *Beginner Mode* - Where the menu option is selected by tapping it with another hand (Figure 3.9).

- *Expert Mode* - Where the menu option is selected by dragging the hand that called the marking menu in the direction of the option (Figure 3.9). One important note is the gesture relaxation: after calling the marking menu with 5 fingers we can relax the gesture to only one finger, in order to select the option with minimal effort.

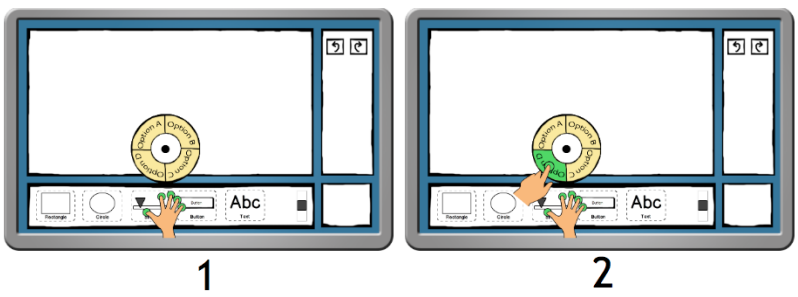

Figure 3.9: Marking Menu - Beginner Mode (Two hands selection) Mockup

Figure 3.10: Marking Menu - Expert Mode (Gesture selection) Mockup

3.4.6 Virtual Keyboard

- **Related To:** US-001, US-003, US-004, US-007, US-021, US-027
- **Description:** Because of the lack of a physical keyboard, and the evident necessity of text input, a virtual keyboard must be available, providing text input. Figure 3.11 shows an example of using the virtual keyboard for renaming an Element Library Group.

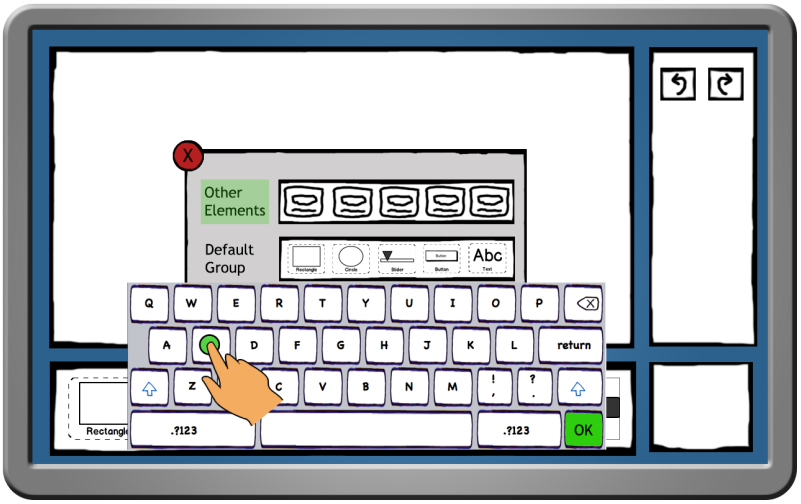

Figure 3.11: Virtual Keyboard - Renaming a Library Group Mockup

3.4.7 Undo Confirmation Dialog

- **Related To:** EPIC-001, EPIC-002, EPIC-003, EPIC-004, EPIC-005
- **Description:** The Undo Confirmation Dialog (Figure 3.12) is used for destructive actions, as an information tool of what happened in the system, and providing an instant Undo for that action.

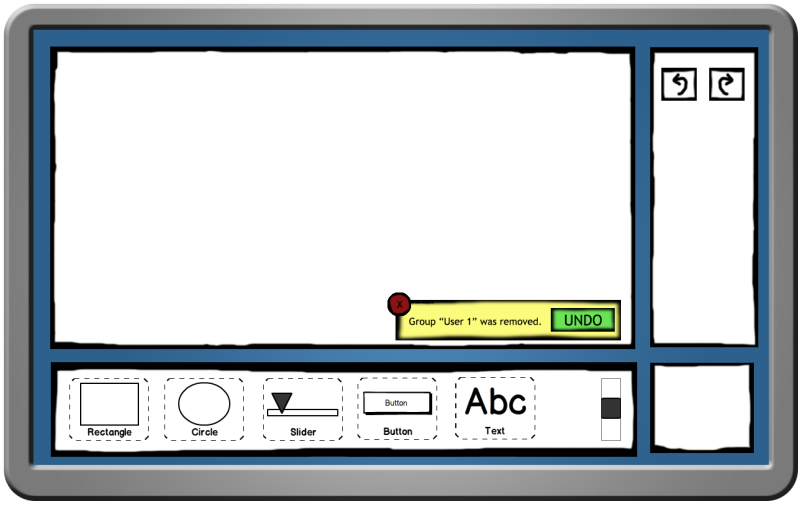

Figure 3.12: Undo Confirmation Dialog Mockup

3.4.8 Layering

- **Related To:** US-008
- **Description:** In order to properly place elements stacked on top of each other, the following layering mechanism is proposed (Figure 3.13 and Figure 3.14).

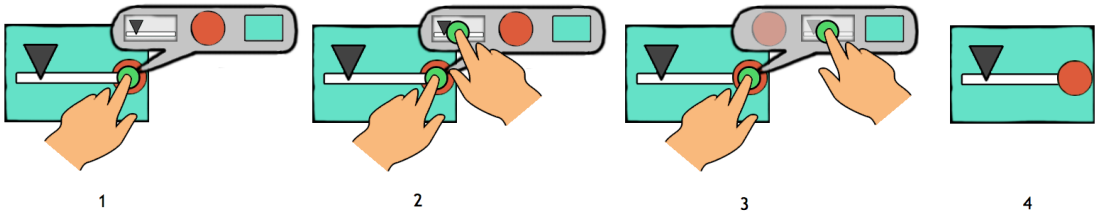

Figure 3.13: Layering Ordering - Reordering Mockup. (1) A long press on an area with various objects pops-up a bubble (2) with the representations of those objects. Another finger drags the representation in the bubble to the desired position(3) effectively reordering the objects (4).

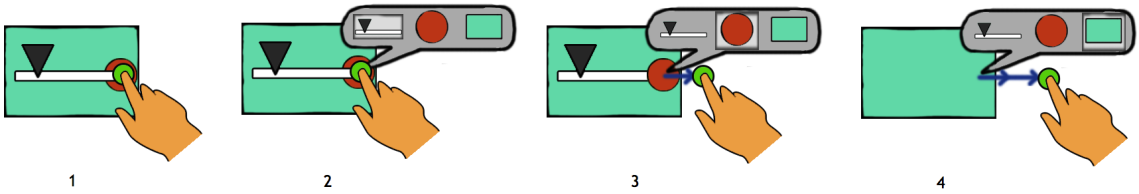

Figure 3.14: Layering Ordering - Bring to Front Mockup. (1) A long press on an area with various objects pops-up a bubble (2) with the representations of those objects. Dragging the finger horizontally will reorder the objects by bringing the selected element to the front

3.4.9 Control Pad

- **Related To:** EPIC-001, EPIC-003, EPIC-006
- **Description:** In order to differentiate the actions of each user in a collaborative environment, each will have a control pad, which is a movable container for the Library and Toolbox (Figure 3.15).

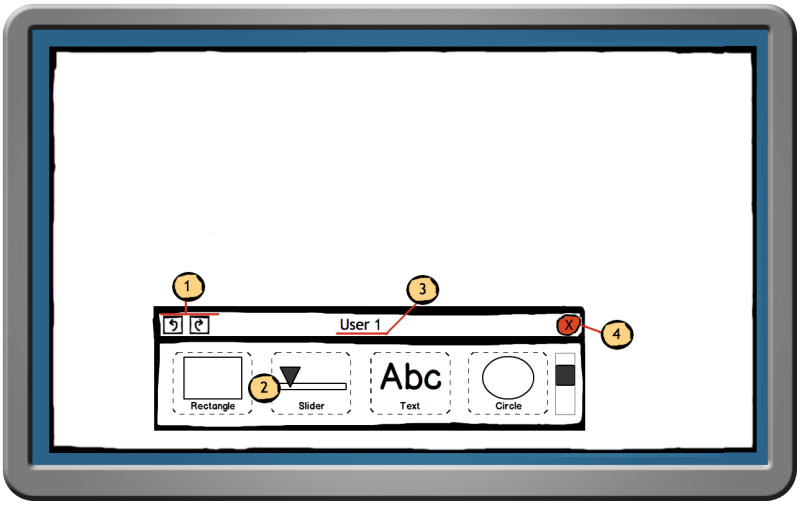

Figure 3.15: Control Pad Mockup. 1) Toolbox; 2) Element Library; 3) User Messages; 4) Dismiss Control Pad.

3.4.10 Create Element

- **Related To:** US-003, US-004
- **Description:** Users can select a grouping of objects or a freehand sketch and drag them to the Element Library to create new elements, so they can use them in the future (Figure 3.16).

3.4.11 Gestural Sketching

- **Related To:** US-005
- **Description:** Elements can be placed on the Designer Area by gestural sketching (for instance, drawing an approximate rectangle or circle places those elements from the Element Library) (Figure 3.17).

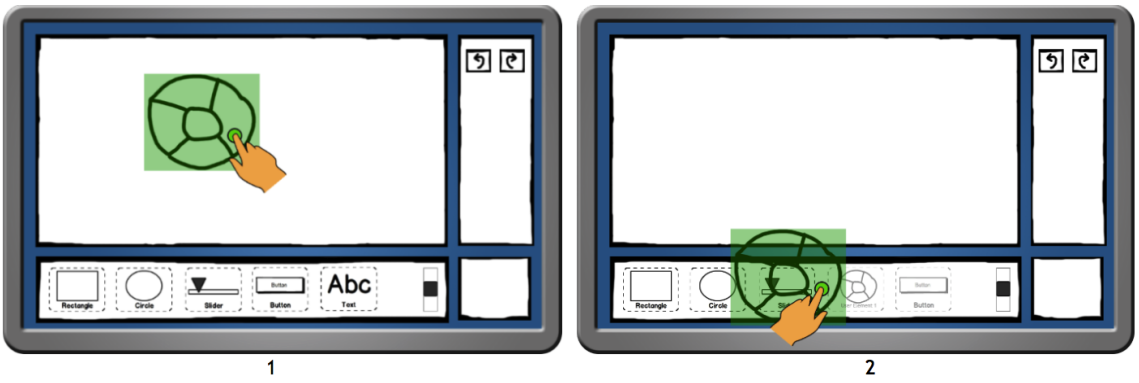

Figure 3.16: Create Element Mockup. The sketch is selected (1) and dragged to the Element Library (2), where the new element is created.

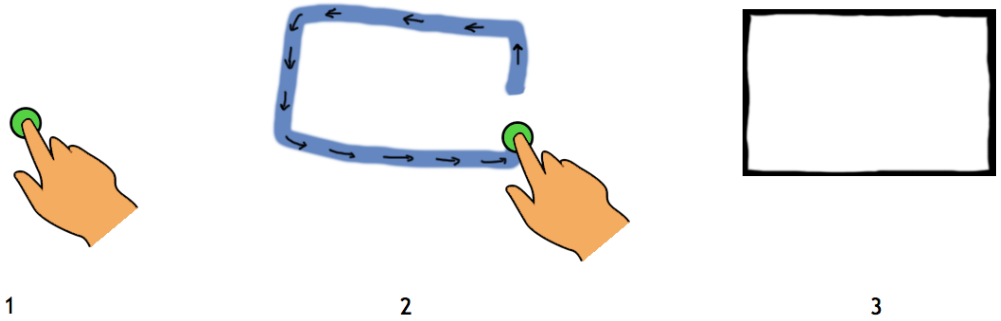

Figure 3.17: Gestural Sketching Mockup. The user places the finger on an empty area in the Designer Area (1) and starts drawing a gesture (2). When the finger is released, the recognized element is placed on the area drawn by the user (3).

3.5 Summary

By analysis of existing prototyping tools and by following design principles applied to NUIs and multi-touch interaction, an initial specification was proposed. This specification assumes a scope of development bigger than the scope of this dissertation, which only focus on developing a functional prototype with a subset of selected user stories.

Two tools were identified as essential for this solution: the *Editor Tool*, for designers and stakeholders, and the *Viewer tool*, for testers. An approach for developing this solution, based on UCD and agile methodologies was conceived. Finally a set of User Stories and mockups explaining relevant concepts of interaction were presented.

Chapter 4 details the architecture that supports the solution specified in this chapter.

Chapter 4

Architecture Overview

In this chapter the architecture of the system is specified. The OCGM interaction style, detailed in Section 2.4.2, was followed as a basis for this architecture. This choice was made because studies [GB10, Sal10] indicated that this interaction style is generic enough to support expansion and foster designer creativity. At the same time it was an opportunity to validate this framework in a real project. Figure 4.1 illustrates the five main components of the domain model of the system:

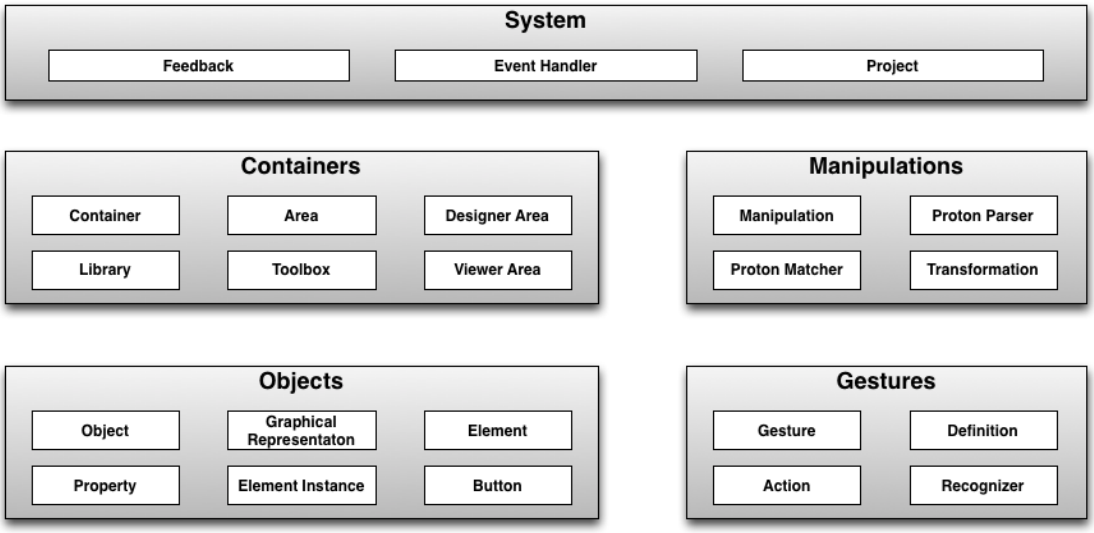

Figure 4.1: Domain Model Overview

- *Objects*, detailed in Section 4.1, which are units of content or data;
- *Containers*, detailed in Section 4.2, for indicating relationships between objects;
- *Gestures*, detailed in Section 4.3, which are symbolic and indirect interactions;

- *Manipulations*, detailed in Section 4.4, for continuous and direct interaction;
- *System*, detailed in Section 4.5, which translates touch input into events recognized as gestures or manipulations, while showing objects and containers as output.

4.1 Objects

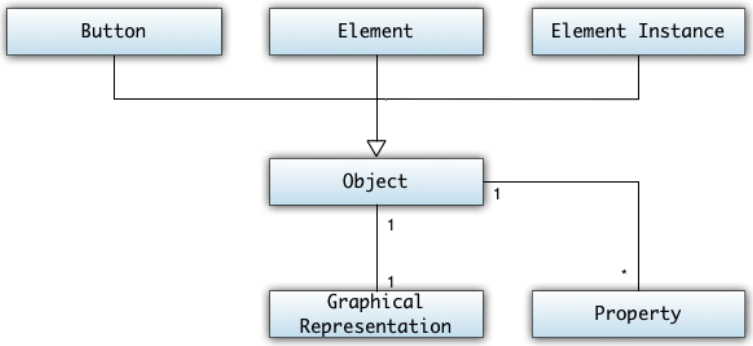

Figure 4.2: Objects - Domain Model Overview

Figure 4.2 shows a diagram of the Objects Domain, and its relationships. *Objects* are the discrete unit of content, providing a common base class for:

- *Elements*, within an Element Library, providing a template for instantiating *Element Instances*;
- *Element Instances*, which are the units with which the Designer can compose a prototype.
- *Buttons*, which are system's controls in the Toolbox, that the Designer can tap to select or trigger an action.

Objects can be composed of several *Properties*, which are attributable values (for instance: Size, Position, Color, Label) and they have a *Graphical Representation*, which provides the visual representation of the Object on the screen.

4.2 Containers

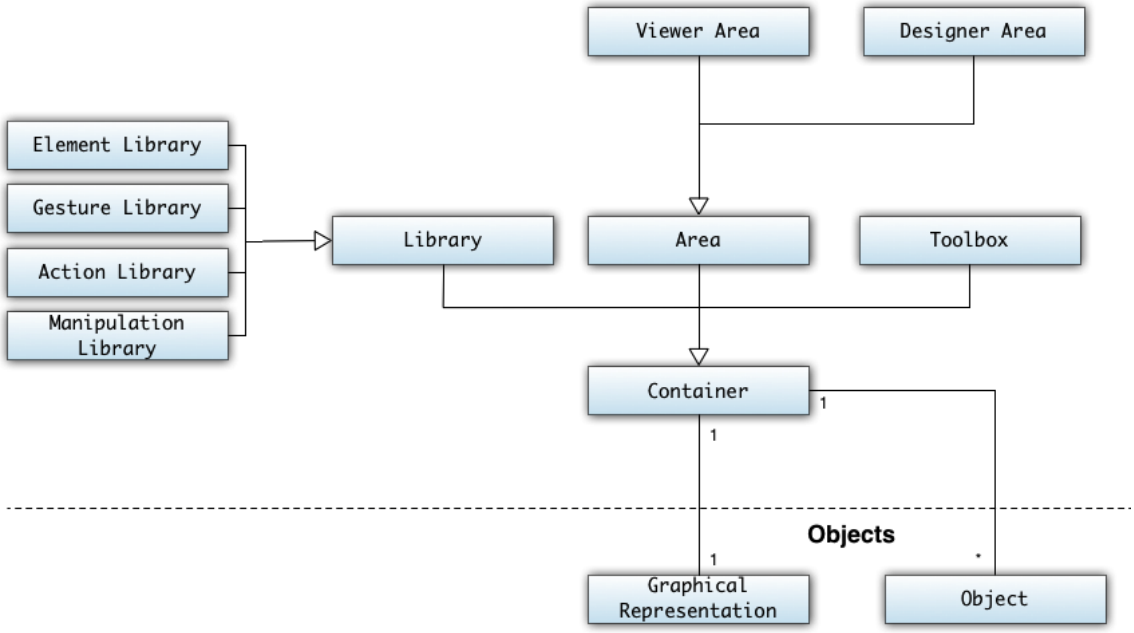

Figure 4.3: Containers - Domain Model Overview

Figure 4.3 shows a diagram of the Containers Domain, and its relationships. *Containers* define relationships between the *Objects*. As such, at its essence, a *Container* is a base class composed by a collection of *Objects*. A *Container* can have a *Graphical Representation* associated, in order to make that relationship visible for the user. There are several kinds of *Containers*:

- *Library*, which divides its objects into groups, with the intention of making a large number of objects easily accessible and browseable (See Section 3.4.3). There are four types of libraries, hinting at the type of objects they contain: *Element Library*, *Gesture Library*, *Action Library* and *Manipulation Library*.
- *Area*, which are placeholders of *Element Instances* (see Section 4.1). In the *Designer Area*, available on the Editor Tool, the *Element Instances* can be manipulated (ie: translated, scaled and rotated) on this area. In the *Viewer Area*, available on the Viewer Tool, these manipulations only occur if the Designer attributes them to each corresponding *Element Instance*.
- *Toolbox*, which is a container of *Buttons* (see Section 4.1), that the Designer can tap for triggering corresponding actions or modes.

4.3 Gestures

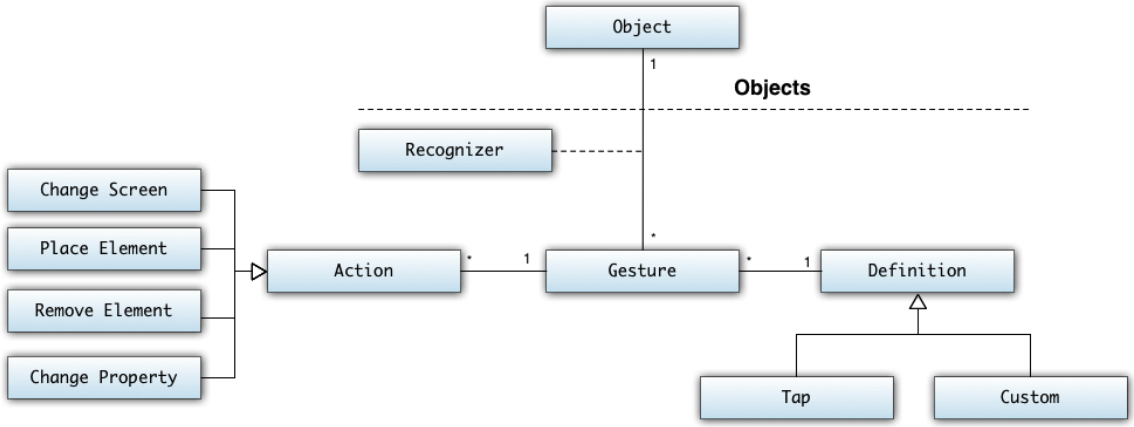

Figure 4.4: Gestures - Domain Model Overview

Figure 4.4 shows a diagram of the Gestures Domain, and its relationships. *Gestures* define symbolic, indirect interactions with *Objects*.

These interactions have a *Definition* associated, for example, the *Tap* Definition, which is a simple Down-Up movement on an *Object*, or a *Custom* Definition, defined by a set of points. For instance, we can define a Rectangle gesture inside the *Designer Area* to place a Rectangle or Div *Element*, or a Cross gesture to remove a specific *Element*.

A *Gesture* also has a sequence of *Actions* associated, which essentially operate on an *Area* (Designer or Viewer). These are preprogrammed actions, such as Change Screen, Place Element, Remove Element or Change Property.

Finally, a *Recognizer* is associated with each *Object*, storing all *Gestures* registered for that *Object*. If a gesture is performed inside the bounding area of an *Object*, its *Recognizer* will select what is the most likely gesture, by running a gesture recognition algorithm.

4.4 Manipulations

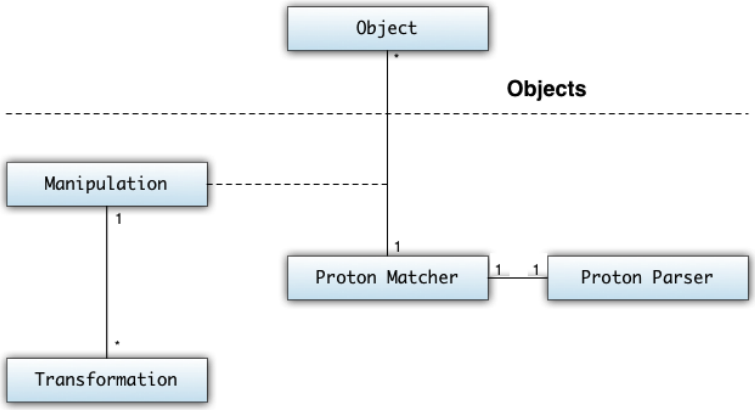

Figure 4.5: Manipulations - Domain Model Overview

Figure 4.5 shows a diagram of the Manipulations Domain, and its relationships. *Manipulation* define continuous, direct interactions with *Objects*, such as Translation, Scaling and Rotation. The Manipulation system was based on the Proton framework [KHD12], which is discussed in more detail in Section 5.3.2. As part of this implementation, a *Proton Matcher*, which has a dictionary of *Manipulation*, sorted by *Object*. A *Manipulation* is defined by a regular expression, parsed by the *Proton Parser*, and a set of *Transformations*, to indicate in which state, and what transformation is applied, to the associated *Object*.

4.5 System

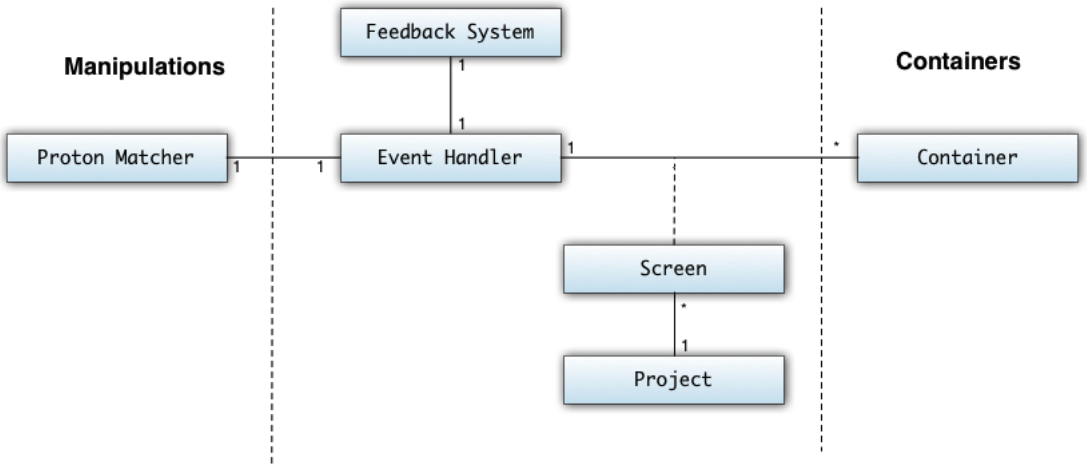

Figure 4.6: System - Domain Model Overview

Figure 4.6 shows a diagram of the System Domain, and its relationships. The most important component of the *System* is the *Event Handler*. The *Event Handler* captures touch events and feeds them to the *Proton Matcher*. The *Event Handler* is where the main *Containers* (such as Area, Libraries and Toolbox) are stored. A *Project* takes care of data persistence in a XML file, loading and saving *Screens*, which are representations of the *Designer Area Container*. Finally, the *Feedback System* component graphically alters *Objects*, *Containers* or the whole *System*, in order to give users visual feedback on their action (for instance, it could add an overlay to an object, to show it is currently being touched).

4.6 Project File

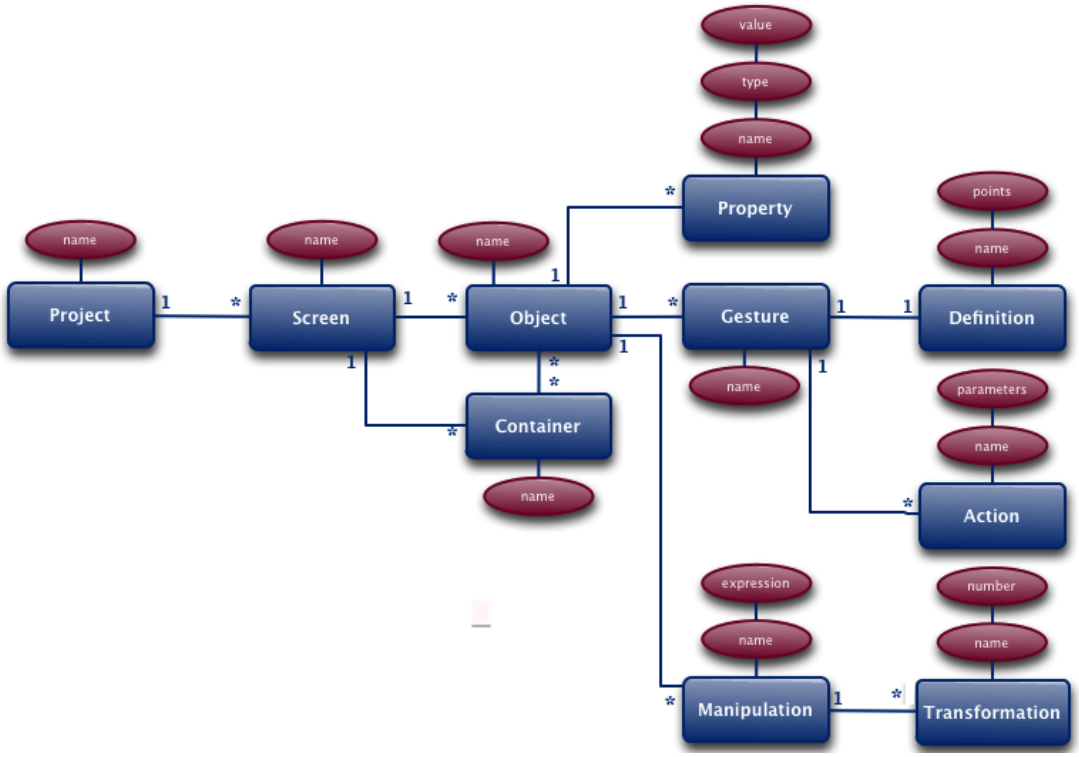

Figure 4.7: Project File - Hierarchical Overview

Figure 4.7 shows a diagram of the Project File, organized by hierarchical relationships (for instance, as an XML file). Each blue ellipse (connected horizontally) represents a node and each wine colored ellipse (connected vertically) represents an attribute of the node it is connected to. The nodes represented are:

- *Project*, the root node, containing information about the project.
- *Screen*, a collection of objects or containers that compose a Screen, which could, for instance, represent a dialog or a window of the interface.

- *Object*, an instance of an Element, that is placed on the Screen. GUI Controls, such as scrollbars, labels, etc., are examples of Objects.
- *Container*, a Grouping of Objects, placed on the Screen to indicate some relationship between them.
- *Property*, represents a property of an Object, such as its position, size and text.
- *Gesture*, which can be attributed to an Object, contains a *Definition* (a set of points) and a list of *Actions*, indicating the sequence of instructions to follow if this gesture is recognized.
- *Manipulation*, defined by a regular expression (as a String) and containing a set of *Transformations* that act upon reaching certain states of the regular expression, as specified by Proton [KHD12].

4.7 Prototypes based on the OCGM Interaction Style

Prototypes created by designers with the tool are based on the OCGM interaction style. As such, not only the architecture of the tool, but also the work-flow of prototyping, is based on this style. This provided an opportunity to test the OCGM as a generic style of interaction, generic enough to prototype novel ways of interaction. As such, the solution specified offers designers:

- *Element Instances*, which are *Objects* that compose an interface;
- *Groupings*, which are *Containers* that define a grouping relationship between objects;
- *Manipulations*, by direct and continuous transformations on objects, by defining user input via Proton Regular Expressions;
- *Gestures*, by indirect gestures on an object (such as cross, square, circle gestures) to cause a sequence of actions;

4.8 Summary

The OCGM Interaction Style aims to present universal foundation metaphors for the NUI paradigm. As such, the architecture specified in this chapter was based in an architecture composed of Objects, Containers, Gestures and Manipulations. This chapter illustrated the domain model of the system, and how each concept is connected. A System module glues these four metaphors in a complete prototyping solution. Finally, a Project File is presented, that explains how the information about a prototype can be stored for data persistence.

Chapter 5 discusses implementation details, that arose from implementing the architecture defined in this chapter, to produce the functional prototype of the solution specified in Chapter 3.

Chapter 5

Implementation Details

Based on the specification presented in Chapter 3 and the architecture specified in Chapter 4, a functional prototype was implemented, in order to act as a proof of concept of the solution, to be validated through user testing.

Some initial implementation decisions had to be taken, such as the technologies to use as a basis for the solution, which are discussed in Section 5.1. This prototype covered a subset of the functionality specified for the whole solution, which is enumerated in Section 5.2.

The most relevant details about the implementation are presented and discussed in Section 5.3. Finally, a case study showcasing the functionality of the prototype is presented in Section 5.4.

5.1 Technologies

The technologies that were used for implementing this functional prototype, as well as the reason for their choice are now presented.

5.1.1 Kivy Framework

The solution was implemented using the Kivy framework [Aut11]. From the list of frameworks analyzed in section 2.5.4, Kivy was chosen because it offered a good documentation, a large user-base, frequent updates and for being cross-platform.

5.1.2 Multi-touch Hardware

For development purposes, the Apple MacBook Trackpad multi-touch emulator provided by Kivy was used. After each use case was implemented, a multi-touch overlay over a 30” screen with 4-finger support was used for more thorough testing. The user tests were conducted in that same screen, mounted horizontally to be operated as a multi-touch tabletop.

5.2 Functional Prototype

As detailed in the work plan (Section 3.2), two milestones of development were completed, in order to produce a functional prototype, that would serve as a basis for further development, and to validate this solution as a proof of concept. In this section, the functional prototype is presented, by listing what functionality was implemented, accompanied by screenshots to illustrate selected cases.

5.2.1 Overview

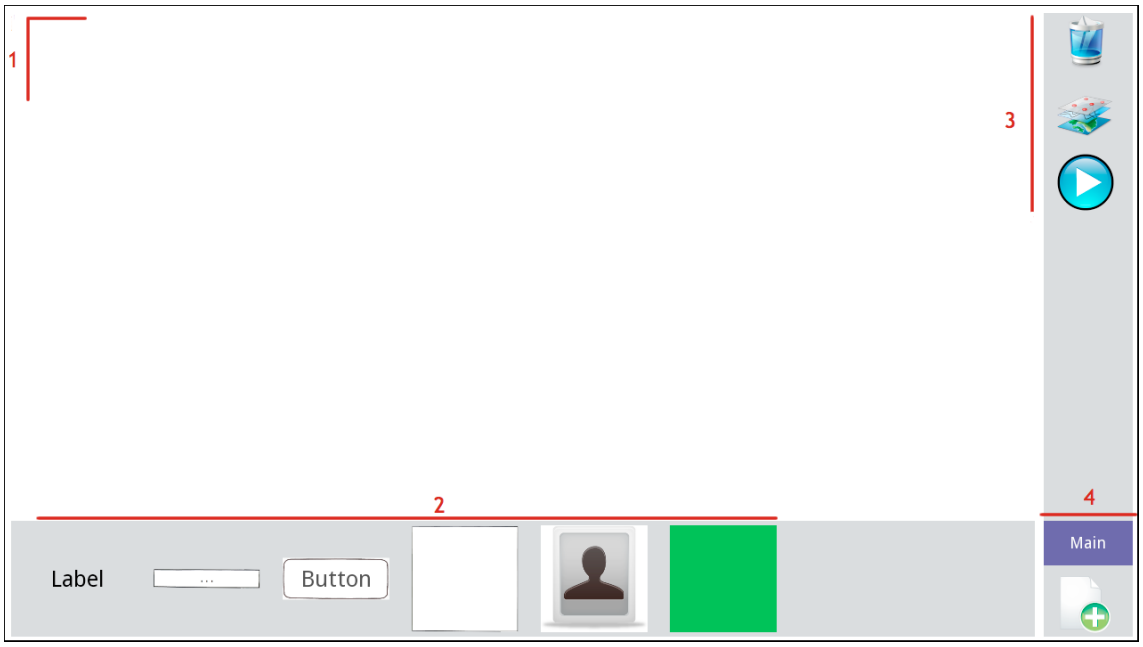

Figure 5.1: Functional Prototype - Overview

Figure 5.1 presents an overview of the Editor Tool. This can be divided into four main areas:

1. *Designer Area*, the canvas where the designer places the elements that compose their prototype.

2. *Element Library*, containing the representations of the elements that can be placed in the Designer Area.

3. *Toolbox*, offering three tools: *Remove Element Instance*, *Send to Back* and *Preview* (from top to bottom)

4. *Screens Area*, where the designer can alternate between screens or create new screens.

5.2.2 Functionality

Table 5.1 lists the functionality implemented in the functional prototype, and the user story covered by it.

Table 5.1: Functional Prototype - Functionality implemented

User Stories Covered	Functionality Implemented	Triggered by
US-005	Element Placement	Dragging an element from the Element Library to the Designer Area.
US-009	Element Removal	Selecting an element instance and tapping the Trash button in the Toolbox.
US-006	Element Translation	Dragging an element instance with one finger.
US-006	Element Scaling	Using a two-finger pinch manipulation.
US-008	Bring to Front	Selecting an element instance.
US-005	Send to Back	Selecting an element instance and pressing the *Send to Back* button on the *Toolbox*.
US-021	Create Screen	Tapping the *Create Screen* button on the *Screens Area*.
US-022	Select Active Screen	Tapping the corresponding screen button on the *Screens Area*.
US-018	Attribute Tap -> Change Screen Connection	Dragging a screen button from the *Screens Area* to an element instance on the *Designer Area*.
US-026	Preview	Tapping the Preview button on the *Toolbox*.

Additional functionality, not specific to a user story, was implemented:

- *Feedback*, the Feedback system was used, with two examples: *Selection Feedback*, a blue border bounding an *Element Instance* to show it is selected, and *Screen Change Attributed*, to show that an *Element Instance* had the Tap -> Change Screen gesture attributed, by connecting a green line between the *Element Instance* and the corresponding *Screen*;
- *Project File Persistence*, an XML file for data persistence was used, based on the Project File specified in Section 4.6

5.3 Core Implementation Details

In this section, the most relevant details about the implementation, the major hurdles surrounding it, and how they were solved are presented. Section 5.3.1 presents a general overview of how the functional prototype was implemented. Section 5.3.2 discusses how the Manipulations were implemented, by applying and extending the *Proton Manipulation Framework*. Section 5.3.3 discusses how Gestures were implemented as a part of the Manipulation system. Section 5.3.4 describes how the *Graphical Representation* of *Elements* was implemented. Finally, given the

importance of feedback in any usable system, the *Feedback* mechanism is presented in Section 5.3.5.

5.3.1 Core Implementation Overview

The architecture specified in Chapter 4 was implemented in Python modules, with each concept of the domain specified as a separate module. An overview of how the system was implemented and integrated with the Kivy Framework is presented in Figure 5.2.

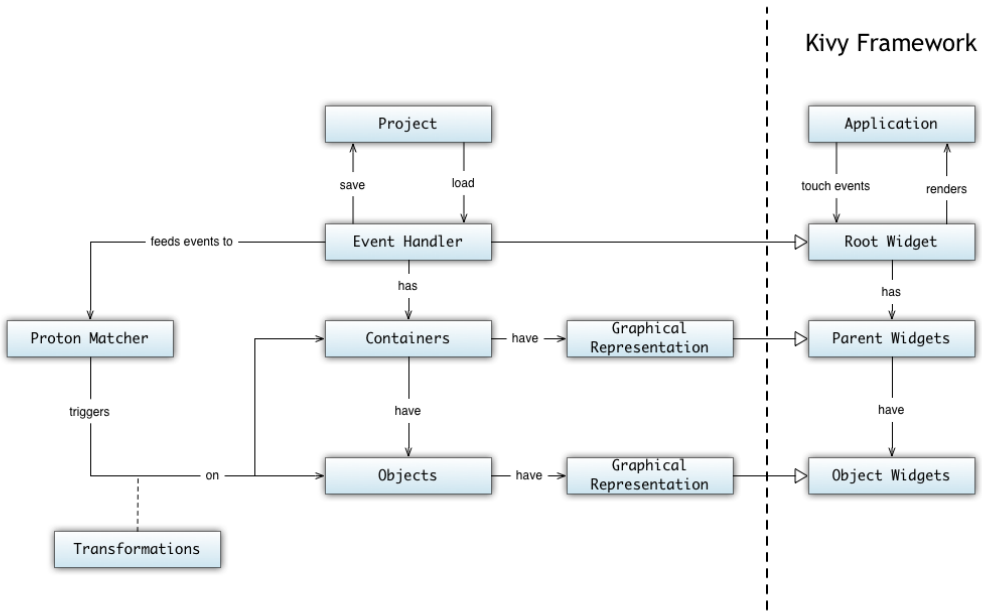

Figure 5.2: Implementation Overview

The *Application* is a *Kivy Application*, which has, as its *Root Widget*, the *Event Handler*. This setup allows for the *Event Handler* to contain all *Containers* of the system. When a *Container* is added to the *Event Handler*, its *Graphical Representation* (a *Kivy Widget*) is added as a child of the *Event Handler*, which renders the *Graphical Representation* of the *Container* and all its children (which are *Objects*).

When a *Project* is loaded, the *Containers*, *Objects*, *Gestures* and *Manipulations* defined in the project file (see Section 4.6) are used to populate the *Event Handler*.

The *Event Handler* intercepts all events received and feeds them to the *Proton Matcher*, which decides the appropriate transformations to apply to *Objects*, according to the *Manipulations* and *Gestures* specified.

5.3.2 Manipulations - Extension of the Proton Manipulation Framework

In order to allow designers to be creative with the kinds of interactions they create for a prototype, a versatile, unrestrained and easy to use gesture representation framework was selected: *Proton* [KHD12]. The representation of gestures as regular expressions is perhaps the most interesting feature of this framework, due to the Gesture Tablature Editor, which is a visual way of creating new gestures, that could easily fit in the Editor Tool.

In the original Proton specification, the manipulation is described as a regular expression, with each symbol having the representation specified in Equation 5.1 [KHD12]:

$$E_{T_{ID}}^{O_{Type}} \tag{5.1}$$

In this representation, *E* is the type of the Touch Event (Down, Move or Up), *TID* is the numbered identifier of each touch point and *OType* is the Object identifier of the object being touched.

A manipulation can be represented as a regular expression of those symbols. For instance, Figure 5.3 presents the three "standard" manipulations on an object, described as *Proton* regular expressions.

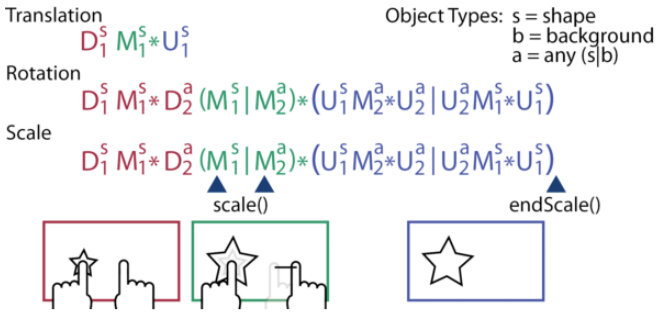

Figure 5.3: Proton regular expressions for translation, rotation and scale manipulation. The thumbnails illustrate the user's actions corresponding to the colored symbols for the scale manipulation. [KHD12]

However, *Proton* had several limitations, mostly due to its recency as a framework. There was no implementation available, only the description of its implementation [KHD12], and the framework was not tested for such a generic environment. There was also an important limitation, the algorithm did not support parallel gestures, which would invalidate a physically collaborative environment for the tool. Despite these shortcomings, the decision of using this framework was adequate and brought an opportunity for the proposal of four improvements to the original Proton framework, which were implemented in the functional prototype. These are:

1. *Hierarchy Support*, which adds hierarchical semantics of containment to the regular expressions. For instance, saying that a manipulation is only applicable if an *Object* belongs to an *Area* or *Group*;

2. *Hovering Support*, which adds hovering support when a object is "picked". For instance, we can add a gesture that erases an *Object* when it is released on top of a *Trash* container, and provide visual feedback when we are over it;

3. *Object Tagged Manipulations*, organizing manipulations by the *Objects* of their first *Touch Down* events, which improved the performance by limiting the number of derivatives analyzed and helped the implementation of *Parallel Manipulation Support*;

4. *Parallel Manipulation Support*, which allowed for multiple gestures to be recognized at the same time, allowing multi-user and multi-gesture support.

5.3.2.1 Hierarchy Representation

The hierarchy would be represented by changing *OType*. Instead of *OType* being a *String* containing the ID of the object it would contain one of the following *String* representations, to indicate hierarchical relationships:

- *Hierarchy*: "DesignerArea > Box001" - Only matches if the touch hits the Box001 and the DesignerArea container;
- *Object*: "Box001" - Matches if the touch hits the object, regardless of container elements;
- *Container*: "DesignerArea" - Matches if the touch hits the container area. A container area could be a fixed value (for instance, a rectangle bounding the Designer Area) or the union of the regions of the several objects that belong to a container (the regions of three objects belonging to a grouping);
- *Empty Area of a Container*: "DesignerArea$" - Only matches if the touch hits the container area and there are no other objects in that point. The $ sign acts as a modifier to indicate this special relationship.

By adding hierarchy support, several additional manipulations can be specified. For instance, we can apply transformations to groups of objects (translate two objects that were selected and, as such, belong to the "selected" container), or specify that an object can only be dropped in an empty area.

5.3.2.2 Hovering Representation

The manipulation symbol can be extended as illustrated in Equation 5.2:

$$E_{T_{ID}}^{O_{Type}-H_{Type}} \tag{5.2}$$

The *OType* indicates the object that is being dragged, while *HType* indicates the object or container above which the object is being hovered (for the *Move Touch Event*) or released (for the *Up Touch Event*).

For instance:

- "Elem001 - Trash" - Matches if Elem001 is hovering above the Trash element.

5.3.2.3 Touch and Hovering Lists

In order to implement hierarchy and hovering, as specified in Sections 5.3.2.1 and 5.3.2.2, two separate lists were created, one indicating the object being touched and its hierarchy (Touch List) and another indicating the elements that are "below" the object selected (Hovering List), if the element is being dragged. As such, if the "Box001" object that belongs to the "DesignerArea" is selected above the "Trash" object that is inside the "Toolbox", the following lists would be created:

- *Touch List* - [DesignerArea, Box001]
- *Hovering List* - [Toolbox, Trash]

We can then test for the appropriate hierarchy and hovering representations defined in Sections 5.3.2.1 and 5.3.2.2.

5.3.2.4 Object Tagged Manipulations

In order to increase performance and allow parallel manipulation support, a dictionary of manipulations, sorted by the objects they act upon, is maintained. The regular expression is analyzed to retrieve the IDs of all object types and the manipulation is associated with those objects. That way, for each object in the touch list (after a touch down), only the manipulations pertaining to that object are added in the derivative list, improving efficiency and enabling the parallel touch-down support, as described in Section 5.3.2.6.

5.3.2.5 Proton Parser

In order to parse the regular expressions, the compiler construction tool PLY [Bea11] was used as a LALR(1) parser. Listing 5.1 presents the BNF grammar for Proton Regular Expressions, which was based on a BNF grammar for standard regular expressions [Rob99].

The result of the parsing is an Abstract Syntax Tree, which is updated whenever a symbol is consumed, implementing a regular expression derivative mechanism [KHD12].

Using this representation, we could translate a translation manipulation that would drag an "Obj" into an "Area", illustrated in Equation 5.3, as "D-1-Obj (M-1-Obj-Area|M-1-Obj)* U-1-Obj-Area":

$$D_1^{Obj}(M_1^{Obj}|M_1^{Obj-Area})^* U_1^{Obj-Area} \tag{5.3}$$

```
Grammar:

Rule 0     S' -> root
Rule 1     root -> re
Rule 2     re -> union
Rule 3     re -> simple_re
Rule 4     union -> re OR simple_re
Rule 5     simple_re -> concatenation
Rule 6     simple_re -> basic_re
Rule 7     concatenation -> simple_re AND basic_re
Rule 8     basic_re -> replication
Rule 9     basic_re -> elementary_re
Rule 10    replication -> elementary_re STAR
Rule 11    elementary_re -> group
Rule 12    elementary_re -> touch
Rule 13    group -> LPAREN re RPAREN
Rule 14    touch -> TOUCH

Tokens:

TOUCH = r'[DMU]-\d-\w+(-\w+)?'
OR = r'\|'
STAR = r'\*'
AND = r'\s'
LPAREN = r'\('
RPAREN = r'\)'
```

Listing 5.1: Proton regular expression gramar

5.3.2.6 Derivative Matching Algorithm

The derivative matching algorithm, that enables the parallel manipulation supported, is presented as pseudo-code based on the Python Programming Language in Listing 5.2.

The *consumeEvent* method checks if the next symbol on the derivative matches the event received and updates the AST containing the regular expression accordingly. Furthermore, it translates the touch ID received into a relative touch ID, registering it in the list of touches for that derivative. It returns:

- *SUCCESS*, indicating that the event was matched for this derivative, but it has not reached yet the final symbol;
- *MATCH*, indicating that the event was matched for this derivative and has reached the final symbol;
- *FAILURE*, indicating that the event was not matched for this derivative.

The *prospect* method checks if the next symbol on the derivative matches the event received.

A *score list* (as a priority queue) is maintained, in order to easily retrieve the transformation with the highest confidence score for execution.

A more detailed, step-by-step explanation of the algorithm is presented here:

1. On each touch down, retrieves the manipulations for each object in the touch list.
2. For each manipulation retrieved, creates a derivative of the manipulation if that manipulation is not already active. This way, a second touch down in a object will not trigger the same manipulation (we assume that parallel touches will not trigger the same manipulation).
3. The active derivatives are then copied, in order to safely remove derivatives after iterating.
4. For each active derivative:

   If the received event type is MOVE or UP, and if the touch ID of the received event is registered on the derivative, it consumes the event. Otherwise, it skips this derivative. This way, we ensure a fast handling of the MOVE and UP events.

   If the received event type is DOWN, it prospects the object type of the next symbol in the derivative. If it matches the symbol touch list, the received symbol is consumed, otherwise it skips the derivative.
5. The state result of consuming the event is then evaluated:

   If state is *SUCCESS*, the derivative is added to the score list.

   If state is *MATCH*, the derivative is added to the score list and the derivative and manipulation is removed from the active derivative and manipulation lists.

   If state is *FAILURE*, the derivative and manipulation is removed from the active derivative and manipulation lists.

```
1
2  if symbol.eventType == DOWN:
3      for object in symbol.touchList:
4          manipulations = objects[object]
5          for manipulation in manipulations:
6              if manipulation not in activeManipulations:
7                  derivative = ManipulationDerivative(manipulation)
8                  activeDerivatives.append(derivative)
9                  activeManipulations.append(manipulation)
10
11 for derivative in activeDerivatives[:]:    # copy list for safe removal during
       iteration
12     state = None
13     if symbol.eventType in [MOVE, UP]:
14         if derivative.hasTouch(symbol.touchId):
15             state = consumeEvent(symbol, touchInfo, derivative)
16         else:
17             continue
18     elif symbol.eventType == DOWN:
19         if derivative.prospect(symbol):
20             state = consumeEvent(symbol, touchInfo, derivative)
21         else:
22             continue
23
24     if state == STATE_SUCCESS:
25         addToScoreList(derivative)
26     elif state == STATE_MATCH:
27         addToScoreList(derivative)
28         activeDerivatives.remove(derivative)
29         activeManipulations.remove(derivative.manipulation)
30
31     elif state == STATE_FAILURE:
32         activeDerivatives.remove(derivative)
33         activeManipulations.remove(derivative.manipulation)
34
35 executeHighestScoreTransformation()
```

Listing 5.2: Derivative matching algorithm

6. Finally, the transformation with highest score is executed.

5.3.2.7 Application of Proton in the Functional Prototype

Although we did not visually implement the Gesture Tablature system, that would allow designers to build their own manipulations, all the manipulations implemented directly in the functional prototype used our implementation of Proton. These manipulations included:

- Element Placement (from the Element Library to the Designer Area)
- Element Instance Translation and Scaling
- Tap -> Screen Change Connection

This system works perfectly for these cases. However, Proton regular expressions that deal with more than three fingers seem to be too complicated to specify by hand. Simplifications should be studied to this language, which could ultimately be enforced in a visual environment, so that the Manipulation creation mechanism is intuitive to the designer.

5.3.3 Gesture Recognition

Although not implemented in the functional prototype, its architecture is prepared for *Gestures*, by offering *Objects* a *façade*, that only recognizes the *Tap* gesture.

As discussed in Section 2.5.3.2, $1 gesture recognizer [WWL07] is a simple, yet effective and accurate gesture recognition system. As such, for a first iteration of the gesture recognition system, this algorithm is a good candidate for implementation. Since Protractor [Li10] is an improvement to the $1 gesture recognizer, a next iteration for implementing this algorithm will be considered as a next step, although the effort/value cost in implementing this improvement should be considered.

A Gesture can be represented as a Manipulation recognized by Proton, given the regular expression specified in Equation 5.4:

$$D_1^{Obj}(M_1^{Obj})^* U_1^{Obj} \tag{5.4}$$

A *ExecuteRecognizerAction*, executed at the UP Touch Event, would analyze all points traced since the DOWN Touch Event, and execute the gesture attributed to the object with highest score (as specified by the $1 recognizer algorithm [WWL07]).

The *Tap* gesture always returns the highest score, meaning that, independently of the motion of the gesture, the *Tap* gesture is always executed. In a future iteration, the MOVE Touch Event would trace the points of the gesture's motion and the *ExecuteRecognizerAction* would take those points and decide the most appropriate gesture to execute.

```
<RectangleGraphicalRepresentation>:
    width: 200
    height: 200
    canvas:
        Color:
            rgb: 0, 0.7, 0
        Rectangle:
            pos: self.x, self.y
            size: self.width, self.height
```

Listing 5.3: Rectangle element definition on Kivy Language

5.3.4 Element's Graphical Representation

The *Kivy Language* was used to define the graphical representation of the elements, as a way to have the visual representation in a separate, easy to edit, file. In future iterations, vectorial graphics formats such as SVG should be explored for implementing this effect. For instance, Listing 5.3 lists the *Kivy Language* definition for the *Rectangle Element*.

When an Element is placed in the Designer Area, a new *Element Instance* is created, with a newly instanced Graphical Representation, cloned from the Element. As such, Elements act as templates for Element Instances.

5.3.5 Feedback

Given the importance of feedback on any kind of system [Nor02], a mechanism was implemented to easily apply visual feedback on objects. Figure 5.4 shows the *Feedback* interface and its relationship with the *Object* class.

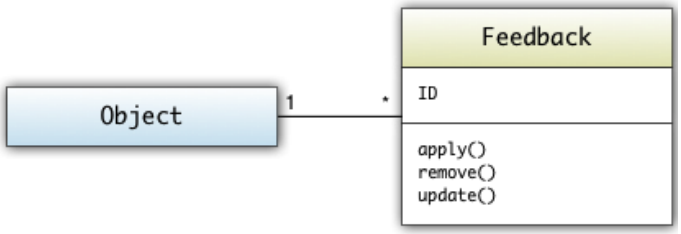

Figure 5.4: Object Feedback

Each subclass of *Feedback* (such as Select, Translation, ValidTransformation, InvalidTransformation) can use the *apply* method to change the representation of an element (for instance, adding a colored semi-transparent layer), *remove* to revert the representation (removing the feedback) and *update* to update the size/position of the feedback, if needed.

A feedback is applied to an element by registering it in a dictionary of feedbacks. That way, if we try to apply the same feedback twice to the same element, it will be ignored. The feedback can

```
1 if self.selectedObject != None:
2     self.selectedObject.removeFeedback(SelectFeedback.ID)
3
4 self.selectedObject = obj # obj is the new selected object
5
6 obj.addFeedback(SelectFeedback())
```

Listing 5.4: Rectangle element definition on Kivy Language

be removed by simply removing the item that has the Feedback *ID* from the dictionary. Listing 5.4 shows how the Select feedback is applied and removed from the selected object.

5.4 Case Study

As an example of use of the functional prototype, and to showcase its functionality, a prototype for a Chat system, with a Login Screen and a Chat Screen, was created with this tool.

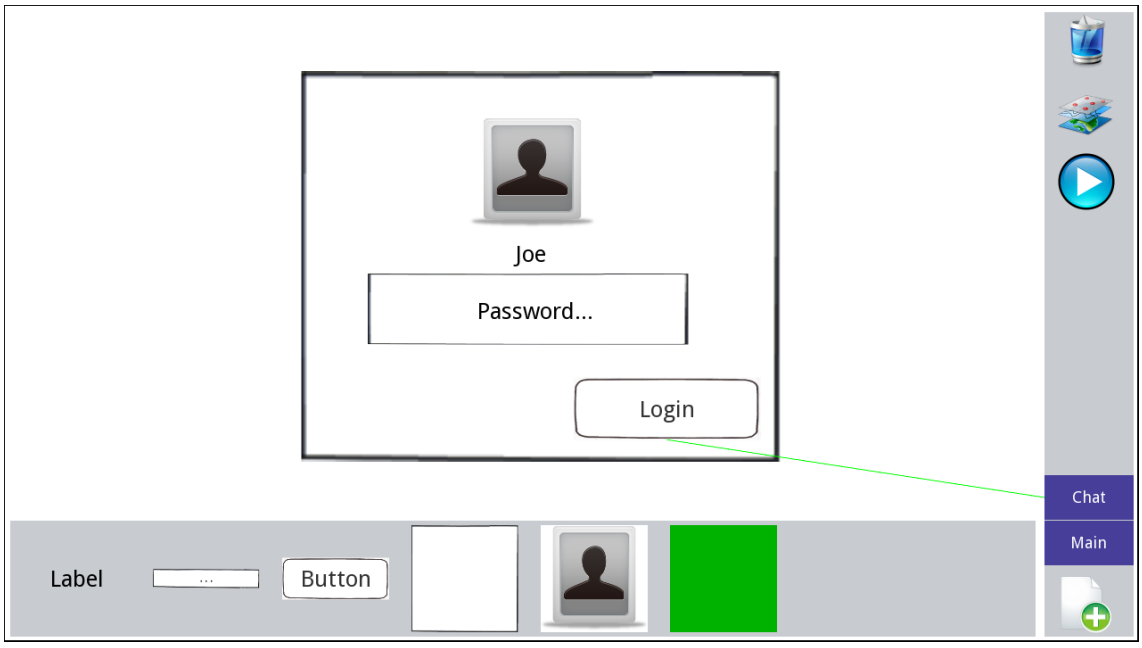

Figure 5.5: Editor Tool - Login Screen

Figure 5.5 presents the Editor tool, with the Main screen active, in which the Login form is created. A green line connecting the Login button and the Chat screen button indicates that, in the Viewer Tool, tapping that button would change to the Chat screen.

Figure 5.6 depicts the Keyboard tool, implemented by the Kivy framework, which appears on the screen when the user places an element that has a label (such as the Label and Button elements). The Keyboard is dismissed by tapping the "X" or "Return" buttons. The label text is

updated as the user types on the Keyboard. The red text appears to indicate the user that it should enter text, and to remain visible if the user places the element below the Keyboard area. Because the Label instance is selected, the Highlight Feedback is applied on that object.

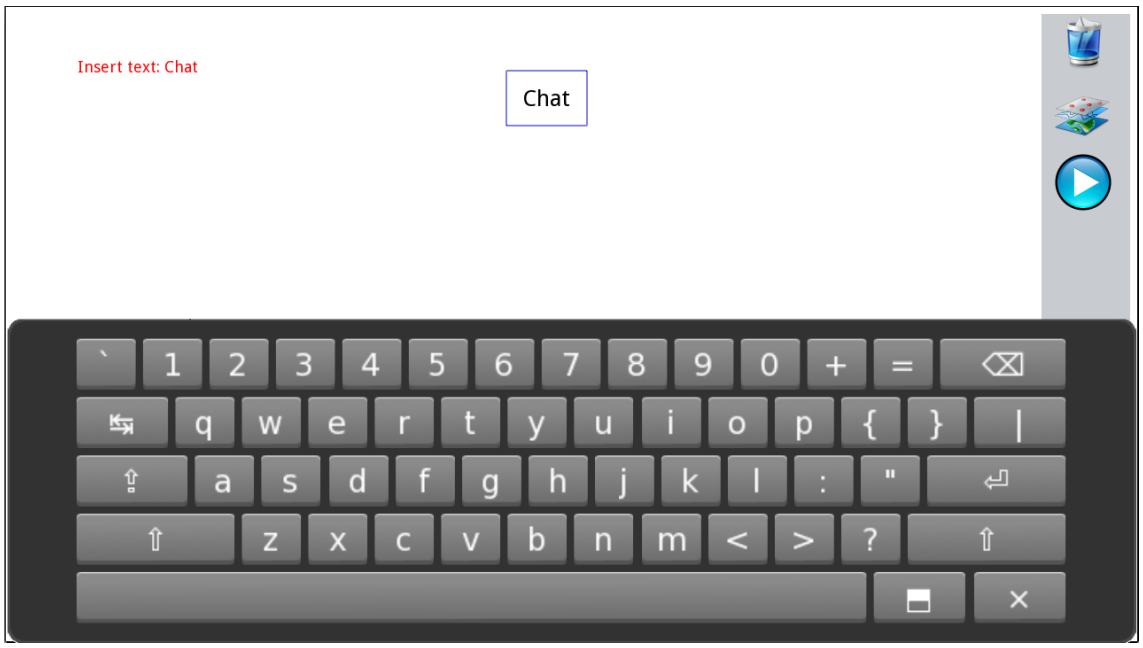

Figure 5.6: Editor Tool - Keyboard

Figure 5.7 presents the Viewer Tool, on the Chat Screen (after tapping the Login button). An Edit button is placed on the top-left corner of the screen, so the user can go back to the Editor Tool.

5.5 Summary

In this chapter, relevant details encompassing the implementation of the functional prototype were presented. The basis technologies in which the prototype was implemented, the *Kivy Framework* and the multi-touch setup, were discussed. The functionality chosen to be implemented in this prototype, the result of two milestones of development was listed.

A series of relevant details about the implementation, the major hurdles surrounding it, and how they were solved were presented, with special focus for the integration of the architecture with the *Kivy Framework*, the implementation and extension of the *Proton Manipulation Framework*, the implementation of the *Gesture Recognition* system, the *Graphical Representation* of Elements and the *Feedback* system.

Finally, a case study showcasing the functional prototype through an example of use, was presented.

This implementation was validated by performing user tests to the functional prototype, which are the focus of Chapter 6.

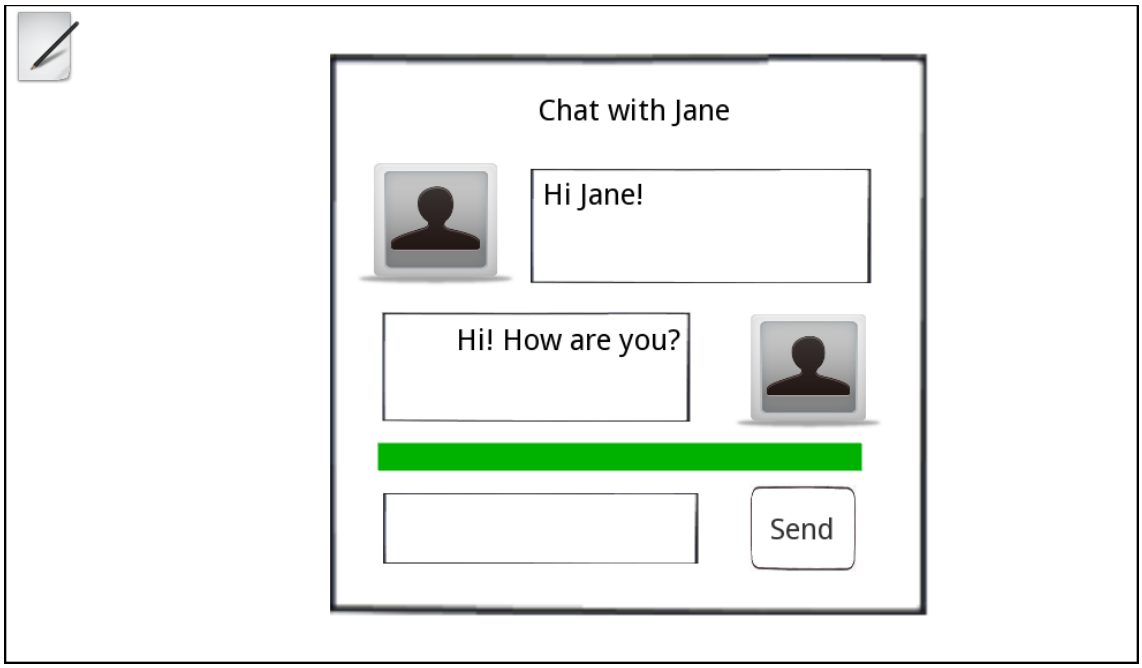

Figure 5.7: Viewer Tool - Chat Screen

Chapter 6

Usability Tests and Validation of the Results

As detailed in Section 3.2, a User-Centered Design approach was followed for the implementation of the solution. As such, a session of usability testing was conducted, in order to accomplish two goals:

1. To validate our solution, as a proof of concept, and to evaluate our initial approach;
2. To gather insights from user observation and from suggestions given by users, to guide further development, as recommended by UCD.

Given the resource constraints, the usability tests conducted were done with few users - seven at total - as recommended by the discount usability movement [Kru06, Nie09]. The methodology followed for user testing was based on qualitative "Thinking out Loud" user observations [Kru06]. Section 6.1 presents the setup and methodology for each session. Section 6.2 profiles the users involved in the test. The results are presented in Section 6.3 and a summary and discussion of these results is presented in Section 6.4.

6.1 User Testing Setup

Each user testing session had two actors:

- the *test facilitator*, conducting the user through the test and annotating its performance and feedback;
- the *user*, the test subject, that will provide the insights.

Each session had from 30 to 45 minutes of duration, structured in five phases:

1. *Introduction and Profiling*, in which the test format was explained and some short questions for assessing the user familiarity with prototyping and multi-touch interaction.

2. *Purpose of the Tool*, after running the functional prototype for the first time, some questions about the initial expectations of the tool, without interacting with it, were placed, in order to better assess the user's mental model.

3. *Tasks*, in which the user was given some creative tasks to accomplish and asked to think out loud as to what he is doing, what he expects to happen and to give his opinion on the way things should happen. This is where the test facilitator must be more observant, in order to spot difficulties of interaction and encourage the user to speak.

4. *Conclusion*, in which the user is asked what he felt about the tool and the whole prototyping in a multi-touch platform experience, and to give feedback on what features he would like to see implemented.

5. *Mockups Evaluation*, in which the user is presented some mockups of future functionality to be implemented and asked to give its opinion about them.

Appendix B presents the full script the test facilitator followed during each of these sessions.

6.2 Users Profile

Seven users were chosen to participate in the usability testing sessions. Since the goal of these sessions was to evaluate user receptivity and their work-flow, by prototyping on these environments, all the users selected had some prototyping experience. The users can be divided in three groups:

- Three students, with a general background in software development and design, still finishing their Master's Degree;

- Three researchers, with a specific background in user experience, interaction design or multi-touch development;

- One interaction designer and web developer, a professional with experience in the industry.

6.3 Results

After compiling and reviewing the annotations, the following insights were gathered, divided by seven categories:

- *Initial Expectations*, gathered by showing the tool to the users and, without them interacting with it, having them say what they think they could do with it (Table 6.1);

- *Positive Reactions*, praise given spontaneously by users about tool's features and the work-flow experience (Table 6.2);

- *Negative Reactions*, difficulties users had, observed or stated by themselves during the tasks (Table 6.3);
- *User Expectations*, these helps us elicit the user's mental model, gathered by observing its failed attempts to cause an effect. For instance, all users tried to drag the object to Trash to remove it (Table 6.4);
- *Suggestions*, given freely by the user, during or after the tasks (Table 6.5);
- *Final Thoughts*, about the experience of prototyping on a multi-touch tabletop (Table 6.6).
- *Mockups Evaluation*, gathered through the user's evaluation of mockups presented during the session (Table 6.7).

Each insight was attributed a frequency, based on the amount of users who stated or induced this insight: High (6-7 users), Medium (3-5 users) and Low (1-2 users).

Table 6.1: Insights gathered during user testing sessions - Initial Expectations

Insights - Initial Expectations	**Frequency**
Positive Expectations	
Identification of at least three main areas: Elements, Canvas, Buttons.	High
Correct identification of the Trash button purpose: to remove an element.	High
Comfortable and simple interface.	Medium
Negative Expectations	
Development of GUI prototypes (forms and windows), but not anything more.	High
The purpose of Buttons on the Toolbox was confusing (lack of text).	High
Incorrect identification of the "Main" screen button as a button to invoke a Menu.	Low
Suggestion for an interactive tutorial showing where to start.	Low

Table 6.2: Insights gathered during user testing sessions - Positive Reactions (High is better)

Insights - Positive Reactions	**Frequency**
Simplicity.	High
Focus on building the prototype (big area).	High
Fast and enjoyable work-flow.	Medium
Good for iterating through several prototype versions	Medium
Virtual Keyboard is accurate and fast.	Low
Attribute Screen Change mechanism.	Low

6.4 Summary and Conclusions

Given the importance of user testing for validation and guiding future development iterations, user tests on the functional prototype were conducted, which were described in this chapter.

Table 6.3: Insights gathered during user testing sessions - Negative Reactions (Low is better)

Insights - Negative Reactions	**Frequency**
Bring to Front whenever an item was touched was annoying.	High
Keyboard should not always be invoked when a text element is placed.	Medium
Lack of feedback on the buttons.	Medium
Attribute Change Screen Action is not clear [1].	Medium
Very difficult to place an object in the middle layer.	Low

[1]Users spent some time experimenting to learn this action, but liked the approach afterwards.

Even with a small sample of users, the goals proposed at the beginning of this chapter were accomplished. All users felt they could prototype on a multi-touch environment and, given a fully functional and robust tool, they would likely prefer this environment to a GUI, mouse-based one. Some of them spontaneously recognized the value of presenting the prototype to stakeholders and immediately brainstorm alternatives with them.

Selecting users with background in interaction design proved to be an extra aid in evaluating the interface, as they identified problems that, although they did not happen to them, could happen with other users, immediately suggesting alternatives for solving the problem. As such, at the end of each session, a discussion about the experience of prototyping and future improvements naturally followed up.

The functional prototype was praised for its simplicity, which should be a key principle to maintain in future iterations. Adding more functionality should not clutter the interface, using contextual environments and scaffolding to reach this effect (see Section 2.4.1).

The layering system should be revised. Bringing the selected object to front whenever it was touched had a very penalizing effect, whenever there was an accidental activation, or when the user only wanted to translate or scale the object without changing the layering order. The model prototyped in Section 3.4.8 should be tested.

For designing interactions, such as attributing Tap -> Change Screen action to an object, a proper system should be designed, in order to allow designers to easily attribute interactions and create new ones. Exploring Proton's Gesture Tablature system to visually create manipulations, the adequacy of using the metaphor of a Library for attributing gestures and zooming interfaces[Ras00, KRR09] (for attributing actions between screens), is a starting point for future iterations.

Providing the system appropriate feedback, with self-revealing gestures and utilization hints, should also be a priority to help the user discover the system in an enjoyable way.

Undo, Grouping and Cloning should be considered for the next milestone of development, since most users felt the need for those features.

There were also some concerns about system's ergonomics. Other alternatives to the horizontal tabletop, the target platform for the Editor tool, should be explored.

Table 6.4: Insights gathered during user testing sessions - User Expectations

Insights - User Expectations	**Frequency**
Drag elements to Trash, to remove them.	High
Double tapping a text element for editing the text.	High
Scaling a Label should resize its font accordingly.	Medium
Tap in an empty area to unselect a selected element.	Medium
Expected to see a list of all elements, sorted by layer ordering, when tapping the Send to Back Button.	Low
Tapping an Element in the Element Library should place the element in the Designer Area.	Low
Horizontal Scrolling on the Element Library would show more elements.	Low

Table 6.5: Insights gathered during user testing sessions - Suggestions

Insights - Suggestions	**Frequency**
Element Cloning (US-012)	High
Undo/Redo (US-013)	High
Element Grouping (US-010)	High
Element Snapping	Medium
Contextual Menu next to an element with Send to Back button	Medium
Lock/Unlock	Low
Cut Object->Screen connections with a finger swipe.	Low
Have the several screens side-by-side, in a zooming interface, and drag an arrow to connect the screens.	Low
Clone Screen	Low

Table 6.6: Insights gathered during user testing sessions - Final Thoughts

Insights - Final Thoughts	**Frequency**
Useful in collaborative contexts.	High
Useful for presenting the prototypes to stakeholders in realtime.	Medium
Could not be as fast as sketching an idea on paper, but is faster than mouse input.	Medium
Enjoyable way of prototyping.	Medium
Could be useful as an interactive/collaborative white-board.	Low
The setup could be a bit tiring after a few hours of work. A white-board or an inclined setup (as an architect table) are options to explore, to improve ergonomics.	Low

Table 6.7: Insights gathered during user testing sessions - Mockups Evaluation

Mockups Set	**Insights - Mockups Evaluation**
Marking Menu	Mixed reactions about using 5 or less fingers. It would need further usability testing for this two options, to evaluate the better one.
List	Most users thought it was a waste of space. Should be compact information.
Element Rotation	Most users were receptive to the 3 finger gesture. Some suggested adding a circle around the object, after placing one finger in the object, and rotate around the circle.
Undo Dialog	Most users thought this dialog unnecessary and obtrusive.
Gestural / Freehand Sketching	Users were enthusiastic about this idea. Some of them felt there should be buttons for each mode. One suggested making Gestural Sketching default and, every time a gesture was recognized, a message would appear that would give the option to use that element, or input the freehand sketch.
Element Creation	They all were receptive to this idea.
Layering	Users were very receptive to this idea. One user suggested showing a preview of the result of each layering option, instead of showing the elements, since it was more intuitive and it would prevent confusion in case there was two elements of the same type.

Chapter 7

Conclusions and Future Perspectives

Prototyping tools are a good candidate to benefit from NUI, multi-touch environments, because of the identified need to unify the best of paper prototypes and software-based prototypes. In this vein of thought, a solution was proposed, after analyzing existing tools and having NUI and multi-touch design principles in mind. This solution was specified, in terms of functionality and architecture, and a functional prototype was conceived, to offer an initial validation of the concept, architecture and to guide future development.

Considering what was done in this dissertation, the goals were fully achieved. The architecture was successfully implemented in a functional prototype and this prototype was tested and validated with users, providing guidelines for future development.

This dissertation provided a series of contributions, summarized in Section 7.1. Nevertheless, there is still room for improvement, so guidelines for future work are suggested in Section 7.2, before closing the document with final remarks in Section 7.3.

7.1 Contributions

Work done in this dissertation led to several contributions to the academic community, that could in the future be applied to the industry. These were:

1. The creation of an initial design of a novel prototyping tool, which is intended to be iterated;
2. The analysis of user tests of a functional prototype of this novel prototyping tool, which validated the solution and provided guidelines for further development;
3. The application of the OCGM interaction style in a generic prototyping environment, helping consolidate its validity as a set of "universal foundational metaphors of interaction";
4. The implementation of the OCGM interaction style as a basis for an architecture of a prototyping tool, further consolidating the previous point;

5. The implementation of the Proton Manipulation Framework, consolidating this recent framework, and proposing several improvements and extensions to this framework;

6. The solution of Proton's documented limitations [KHD12], such as parallel manipulation and trajectory support (gesture support).

7.2 Future Work

This dissertation assumes further development of the solution, by doing future iterations over the results gathered from the user testing presented in Chapter 6. This section provides guidelines into what should be done in future iterations.

The most important limitations with the interface in the functional prototype were the lack of Undo, Cloning, Grouping and the Layering mechanism. It is essential that these are addressed in the future iterations, in order to focus the users' attention in other issues.

When Freehand/Gestural Sketching is implemented and validated by designers, a special test should be conducted to assess the speed, productivity, enjoyment and final quality of prototyping in this tool versus paper prototypes versus software prototypes.

The Interaction Design mechanism, where the designers can attribute and design their own gestures and manipulations, should be properly designed, in order to be sufficiently powerful for designing different, novel kinds of interaction, while being simple and enjoyable enough to follow NUI design principles. Zoomable interfaces and the implementation of Proton Gesture Tablature are suggestions to be explored.

Collaboration with multiple designers should be explored in future iterations, in order to assess the best way to promote collaboration with this solution.

When the tool is robust enough, tests with stakeholders should be done, possibly exploring the appropriateness of interactive white-boards, and how can stakeholders help shape an existing prototype and participate in the prototype design process.

Implementing the Viewer tool for mobile environments, such as smartphones and tablets, should be considered as a next step, as they are the most common multi-touch devices available to consumers.

Finally, the ergonomics of the tool should be studied, in order to have a tool where the user feels comfortable to use for long periods of time. Vertical interactive white-boards and inclined tables are valid alternatives to be explored.

7.3 Final Remarks

Concluding this work, we believe the solution here proposed to be novel, creative and enjoyable, with the potential for replacing conventional prototyping tools and have industry adherence. We believe that, with multi-touch systems becoming increasingly available to the consumers, there is

an opportunity for this tool to be marketable, with interaction designers and software development companies in mind.

By developing an initial design, testing a functional prototype and gathering insights for further development, we consider that all goals for this dissertation were fulfilled. There is room for further improvements, as discussed in the previous section, which should happen in an iterative, user-focused process.

We also hope that the NUI study conducted for this dissertation and its application, inspires other designers to develop their applications using NUI principles where they find applicable.

Appendix A

User Stories

This section presents the user stories specified for the solution proposed. It is important to notice that, given the scope of this dissertation, only a limited set of user stories were implemented and not everyone was fully implemented. However, the user stories here presented focus on specifying a complete system, which would be implemented as a fully-featured solution for prototyping, given the appropriate resources.

The user stories are divided into six epics for the Editor Tool: (Element Library, Designer Area, Toolbox, Interaction Design, Project Management and User Control) and one epic for the Viewer Tool.

A.1 Element Library

- **Reference:** EPIC-001
- **Description:** As a Designer, I want to have easy access to a library of elements. This library acts as a pallette, ordered by groups of elements, providing a visual environment for discovering the elements available for prototyping the interface.

A.1.1 Element Library Organization

- **Reference:** US-001
- **Description:** As a Designer, I want to create or remove groups of elements in the library, change the element order inside a group, move elements between groups and change the group order, so that I can organize the element library.

A.1.2 Element Library Browsing

- **Reference:** US-002
- **Description:** As a Designer, I want to browse the elements on the element library, in order to quickly locate the one I want.

A.1.3 Create Composite Element

- **Reference:** US-003
- **Description:** As a Designer, I want to create a new element type that is the grouping of selected elements in the prototype, and add the new element to my library of elements.

A.2 Designer Area

- **Reference:** EPIC-002

  Description: As a Designer, I want to have a WYSIWYG area where I can create my prototype and edit it by directly manipulating the elements I place from the Element Library.

A.2.1 Freehand Sketching

- **Reference:** US-004
- **Description:** As a Designer, I want to freely add sketches to the prototype, which I can add to the element library as a new element.

A.2.2 Element Placement

- **Reference:** US-005
- **Description:** As a Designer, I want to place an element in a specific location of the Designer Area, in order to build a prototype using elements from the Element Library.

A.2.3 Element Instance Direct Manipulation

- **Reference:** US-006
- **Description:** As a Designer, I want to manipulate an instance of an element (in the Designer Area), so that I can move its position, scale its size or rotate its orientation by directly touching them.

A.2.4 Element Instance Properties

- **Reference:** US-007
- **Description:** As a Designer, I want to change the properties of an element instance, so that I can customize its appearence or behaviour.

A.2.5 Element Instance Layering

- **Reference:** US-008
- **Description:** As a Designer, I want to change the way elements are stacked on top of each other in the Designer Area.

A.2.6 Remove Element Instance

- **Reference:** US-009
- **Description:** As a Designer, I want to remove elements no longer needed in the Designer Area.

A.2.7 Group Selection

- **Reference:** US-010
- **Description:** As a Designer, I want to select multiple elements, in order to apply operations or transformations to them.

A.2.8 Grid Navigation

- **Reference:** US-011
- **Description:** As a Designer, I want to navigate through the Designer Area, being able to zoom in and out, while maintaining an overview of the whole screen being designed.

A.2.9 Clone Elements

- **Reference:** US-012
- **Description:** As a Designer, I want to clone one or more selected elements, in order to quickly add repetitive elements to my prototype.

A.3 Toolbox

- **Reference:** EPIC-003
- **Description:** As a Designer, I want to have access to a range of tools that provide me with the visible commands I can use to manipulate the prototype, while being contextually relevant and helpful.

A.3.1 Undo/Redo

- **Reference:** US-013
- **Description:** As a Designer, I want my actions to be easily revocable, in order to have the freedom to experiment.

A.3.2 Alignment/Distribution

- **Reference:** US-014
- **Description:** As a Designer, I want to align or distribute elements relatively to each other.

A.3.3 Lock/Unlock

- **Reference:** US-015
- **Description:** As a Designer, I want to lock specific elements, in order to prevent accidental manipulations from me or other designer.

A.4 Interaction Design

- **Reference:** EPIC-004
- **Description:** As a Designer, I want to attribute existing gestures and manipulations to elements in the Designer Area, and define the actions triggered by each. I also want to create new types of gestures and manipulations, that will be stored in a gesture/manipulation library, in order to prepare my prototype for novel ideas of interaction.

A.4.1 Gesture Definition Library

- **Reference:** US-016
- **Description:** As a Designer, I want to browse a library of gesture definitions (ie, a tap on a button, or drawing a circle on a specific element), for creating indirect interactions applied to element instances (see Gestures in Section 2.4.2).
- *Note:* This should be similar to the Element Library defined in EPIC-001

A.4.2 Action Library

- **Reference:** US-017
- **Description:** As a Designer, I want to browse a predefined or user-defined library of actions, for creating interactions applied to element instances, such as Change Screen after tapping a button.

- *Note:* This should be similar to the Element Library

A.4.3 Attribute Gesture

- **Reference:** US-018
- **Description:** As a Designer, I want to attribute a gesture from the Gesture Library and a sequence of actions from the Action Library to an element.

A.4.4 Manipulation Library

- **Reference:** US-019
- **Description:** As a Designer, I want to have a library of predefined or user-defined manipulations, such as one-finger translation, pinch-to-zoom, etc.
- *Note:* This should be similar to the Element Library defined in EPIC-001

A.4.5 Attribute Manipulation

- **Reference:** US-020
- **Description:** As a Designer, I want to attribute a manipulation from the Manipulation Library to an element.

A.5 Project Management

- **Reference:** EPIC-005
- **Description:** As a Designer, I want to manage a prototype project, which is a collection of different screens. These screens can be created or removed from the project. Only one screen at a time can be edited - the active screen.

A.5.1 Create/Remove Screen

- **Reference:** US-021
- **Description:** As a Designer, I want to create or remove screens from a project.

A.5.2 Select Active Screen

- **Reference:** US-022
- **Description:** As a Designer, I want to select which screen I want to edit.

A.6 User Control

- **Reference:** EPIC-006
- **Description:** As a Designer, I want to have my own Control Pad, with a personal library and toolbox, in order to have multiple users physically interacting in a collaborative way, selecting multiple elements and tools, while editing on the same Designer Area.

A.6.1 Create New Control Pad

- **Reference:** US-023
- **Description:** As a Designer, I want to invoke a new Control Pad for my personal editing on the Designer Area.

A.6.2 Move Control Pad

- **Reference:** US-024
- **Description:** As a Designer, I want to move and rotate my Control Pad, in order to better position myself around the multi-touch tabletop.

A.6.3 Dismiss Control Pad

- **Reference:** US-025
- **Description:** As a Designer, I want to dismiss my Control Pad when I am no longer working on the prototype.

A.7 Viewer Tool

- **Reference:** EPIC-007
- **Description:** As a Viewer, I want to be able to focus my attention on the prototype created by the designer, while having a tool for providing feedback.

A.7.1 Start Prototype

- **Reference:** US-026
- **Description:** As a Viewer, I want to be able to see a prototype in action, interact with it and repeat the whole experience afterwards.

A.7.2 Annotate Prototype

- **Reference:** US-027
- **Description:** As a Viewer, I want to annotate a prototype, in order to give designers feedback for future iterations.

Appendix B

Usability Testing Script

B.1 Introduction

B.1.1 Purpose of the test

You have been invited for this session to help us test a new prototyping tool, which is still a work-in-progress, which we want to improve and validate some initial ideas. It is important to notice that it is the tool that is being tested and not you, there's nothing you can do wrong here. First I'll ask some questions about you.

B.1.2 User profile

- Age:
- Occupation:
- Education:
- Prototyping experience:
- Prototyping tools used:
- Multi-touch experience:

We will start by giving you some tasks for you to accomplish using this tool, while explaining your thought process aloud. We want to analyze your interaction with the tool, during your creative process, so there are no right or wrong answers, all your feedback will be valuable. It is the tool that is being tested, not you! If you feel constrained by the tool, or if you feel something could be done differently, we would be glad to hear it from you. We can't give any answers or discuss things with you during the tasks, but we will gladly do it afterwards.

B.2 Purpose of the Tool

- What is your general impression on the tool?
- What do you think you can do with it?

B.3 Tasks

B.3.1 Login Form

1. Develop the prototype for a Login Form, which asks for the user to input its Username and Password. It should have a Login button.
2. When the user clicks the Login button in the viewer, it should show a message in a green background saying success.
3. Be creative! Play with the tool. When you feel the result is right tell me, to advance to the next task.

B.3.2 Chat

1. Use the previous success screen to develop a prototype for a Chat form (only structural). It should have a picture of the user, its name, an area with previous messages (conversation) and an area to enter a new message.
2. Be creative!

B.4 Conclusion

- What did you think about the tool?
- Did you enjoy the concept and experience of prototyping in a multi-touch environment?
- What would you like to see as a feature?

B.5 Mockups Evaluation

- What do you think of the marking menu? Would you expect it to appear on long press with one finger? What about 5 fingers?
- Would you prefer a List of attributes to appear next to the corresponding object or in a fixed place of the editor?
- What do you think about rotating elements using 3 fingers?

- What would you think about an Undo Dialog, that would appear when a destructive action occurs (such as removing an object)? Would you think it obtrusive?
- What about if you could draw symbols for placing existing elements?
- What about creating your own elements by grouping existing elements, or freehand sketching new ones?
- What about layering, did you had any problem with this simple layering method? What do you think about the prototyped method?

References

[ABB+01] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre. Recent advances in augmented reality, 2001.

[AGWF11] Michelle Annett, Tovi Grossman, Daniel Wigdor, and George Fitzmaurice. Medusa: a proximity-aware multi-touch tabletop. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 337–346. ACM, 2011.

[Aut11] The Kivy Authors. Kivy documentation. `http://www.kivy.org/docs/`, 2011.

[AW10] Lisa Anthony and J.O. Wobbrock. A lightweight multistroke recognizer for user interface prototypes. In *Proceedings of Graphics Interface 2010*, pages 245–252. Canadian Information Processing Society, 2010.

[BBvB+01] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development. `http://www.agilemanifesto.org/`, 2001.

[Bea11] David Beazley. Ply (python lex-yacc). `http://www.dabeaz.com/ply/`, 2011.

[BGBG95] Ronald Baecker, Jonathan Grudin, William Buxton, and Saul Greenberg. *Readings in Human-Computer Interaction: Toward the Year 2000*. Morgan Kaufmann, 1995.

[BPF09] Davide Bolchini, Diego Pulido, and Anthony Faiola. "Paper in Screen" Prototyping: An Agile Technique to Anticipate the Mobile Experienc. *interactions*, 6(6):29–33, 2009.

[Bux88] William Buxton. The Natural Language of Interaction: A Perspective on Non-Verbal Dialogues. *INFOR: Canadian Journal of Operations Research and Information Processing*, 26(4):428–438, 1988.

[Bux90] William Buxton. Smoke and Mirrors. *Byte Magazine*, 15(7):205–210, 1990.

[Bux10] Bill Buxton. A Touching Story : A Personal Perspective on the History of Touch Interfaces Past and Future Lost Along the Way. *Society for Information Display (SID) Symposium Digest of Technical Papers*, 41(1):444–448, 2010.

[Bux11] Bill Buxton. Multi-Touch Systems that I Have Known and Loved. `http://billbuxton.com/multitouchOverview.html`, 2011.

[CH10] Adam S. Carter and Christopher D. Hundhausen. How is User Interface Prototyping Really Done in Practice? A Survey of User Interface Designers. *2010 IEEE*

Symposium on Visual Languages and Human-Centric Computing, pages 207–211, September 2010.

[Cho06] M Cho. A new gesture recognition algorithm and segmentation method of Korean scripts for gesture-allowed ink editor. *Information Sciences*, 176(9):1290–1303, May 2006.

[Coh04] Mike Cohn. *User Stories Applied: For Agile Software Development*. Addison Wesley, 2004.

[CRC07] Alan Cooper, Robert Reimann, and David Cronin. *About Face 3: The Essentials of Interaction Design*. Number 3. Wiley, 2007.

[DDV+10] Jan Derboven, D. De Roeck, Mathijs Verstraete, David Geerts, J. Schneider-Barnes, and K. Luyten. Comparing user interaction with low and high fidelity prototypes of tabletop surfaces. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*, pages 148–157. ACM, 2010.

[DDW99] Mary Davidson, Laura Dove, and Julie Weltz. Mental Models and Usability. `http://www.lauradove.info/reports/mentalmodels.htm`, 1999.

[DML+05] Steven Dow, B. MacIntyre, Jaemin Lee, Christopher Oezbek, J.D. Bolter, and Maribeth Gandy. Wizard of Oz support throughout an iterative design process. *Pervasive Computing, IEEE*, 4(4):18–26, 2005.

[EWH07] John Greer Elias, Wayne Carl Westerman, and Myra Mary Haggerty. Patent US7840912 - Multi-touch Gesture Dictionary, 2007.

[FWSB07] Clifton Forlines, Daniel Wigdor, Chia Shen, and Ravin Balakrishnan. Direct-touch vs. mouse input for tabletop displays. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '07*, page 647, 2007.

[GB10] Ron George and Joshua Blake. Objects, Containers, Gestures, and Manipulations: Universal Foundational Metaphors of Natural User Interfaces. In *CHI 2010*, 2010.

[GCG10] M.T. Gorg, Michael Cebulla, and S.R. Garzon. A framework for abstract representation and recognition of gestures in multi-touch applications. In *Advances in Computer-Human Interactions, 2010. ACHI'10. Third International Conference on*, pages 143–147. IEEE, 2010.

[GJS10] Jens Gerken, HC Jetter, and Toni Schmidt. Can touch get annoying? *Conference on Interactive*, pages 257–258, 2010.

[GN96] Don Gentner and J. Nielsen. The anti-Mac interface. *Communications of the ACM*, 39(8):70–82, 1996.

[HB06] Eva Hornecker and Jacob Buur. Getting a grip on tangible interaction. In *Proceedings of the SIGCHI conference on Human Factors in computing systems CHI 06*, page 437. ACM Press, 2006.

[HBC+92] Thomas Hewett, Ronald Baecker, Stuart Card, Tom Carey, Jean Gasen, Marylin Mantei, Gary Perlman, Gary Strong, and William Verplank. *ACM SIGCHI Curricula for Human-Computer Interaction*. ACM, New York, USA, 1992.

[HC11] Uta Hinrichs and Sheelagh Carpendale. Gestures in the wild: studying multi-touch gesture sequences on interactive tabletop exhibits. *Proceedings of the 2011 annual conference*, pages 3023–3032, 2011.

[Hei09] Ethan Hein. The desktop metaphor is, like, so five minutes ago. `http://www.ethanhein.com/wp/2009/the-desktop-metaphor-is-like-so-five-minutes-ago/`, 2009.

[Hig09] Jim Highsmith. *Agile Project Management: Creating Innovative Products*. Addison-Wesley Professional, 2009.

[HKSBM11] A. Hosseini-Khayat, T. Seyed, C. Burns, and F. Maurer. Low-fidelity prototyping of gesture-based applications. In *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*, pages 289–294. ACM, 2011.

[HMD08] Eva Hornecker, Paul Marshall, and NS Dalton. Collaboration and interference: awareness with mice or touch input. *Proceedings of the 2008*, pages 167–176, 2008.

[Hut10] Peter Hutterer. Mpx: Multi-pointer x. `http://wearables.unisa.edu.au/projects/mpx/`, 2010.

[JDM00] A.K. Jain, R.P.W. Duin, and J. Mao. Statistical Pattern Recognition: A Review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(1):4–37, 2000.

[JGH+08] RJK Jacob, Audrey Girouard, LM Hirshfield, M.S. Horn, O. Shaer, E.T. Solovey, and J. Zigelbaum. Reality-based interaction: a framework for post-WIMP interfaces. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 201–210. ACM, 2008.

[JGR10] Hans-christian Jetter, Jens Gerken, and Harald Reiterer. Natural user interfaces: Why we need better model-worlds, not better gestures. In *-Natural User Interfaces*, 2010.

[JMRW01] S Jaeger, S Manke, J Reichert, and A Waibel. Online handwriting recognition: the NPen++ recognizer. *International Journal on Document Analysis and Recognition*, 3(3):169–180, 2001.

[Joh10] Jeff Johnson. *Designing with the Mind in Mind*. Morgan Kaufmann, 2010.

[KAD09] Kenrick Kin, Maneesh Agrawala, and Tony DeRose. Determining the benefits of direct-touch, bimanual, and multifinger input on a multitouch workstation. In *Proceedings of Graphics Interface 2009*, pages 119–124. Canadian Information Processing Society, 2009.

[Kai11] Johanna Kaipio. *Usability in Healthcare: Overcoming the Mismatch Between Information Systems and Clinical Work*. Aalto University, Helsinki, 2011.

[Kal10] Martin Kaltenbrunner. TUIO. `http://www.tuio.org/`, 2010.

[KBBC05] Martin Kaltenbrunner, Till Bovermann, Ross Bencina, and Enrico Costanza. TUIO: A protocol for table-top tangible user interfaces. In *Neuroinformatics*. Citeseer, 2005.

[KHD12] Kenrick Kin, B Hartmann, and T DeRose. Proton: Multitouch Gestures as Regular Expressions. In *CHI'12*, 2012.

[KJ06] Clif Kussmaul and Roger Jack. User interface prototyping: tips and techniques. *Journal of Computing Sciences in Colleges*, pages 188–190, 2006.

[KKFW10] Dietrich Kammer, Mandy Keck, Georg Freitag, and Markus Wacker. Taxonomy and Overview of Multi-touch Frameworks: Architecture, Scope and Features. In *Proc. of Workshop on Engineering Patterns for Multi-Touch Interfaces, Berlin, Germany*, 2010.

[KM10] S.H. Khandkar and Frank Maurer. A language to define multi-touch interactions. In *ACM International Conference on Interactive Tabletops and Surfaces*, pages 269–270. ACM, 2010.

[Kra10] Sven Kratz. Natural User Interfaces in Mobile Phone Interaction. *Interfaces*, pages 1–6, 2010.

[KRR09] W.A. König, R. Rädle, and H. Reiterer. Squidy: a zoomable design environment for natural user interfaces. In *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*, pages 4561–4566. ACM, 2009.

[KRR10] Werner a. König, Roman Rädle, and Harald Reiterer. Interactive design of multimodal user interfaces. *Journal on Multimodal User Interfaces*, 3(3):197–213, February 2010.

[Kru06] Steve Krug. *Don't Make Me Think!* New Riders Publishing, 2nd edition, 2006.

[Kur93] Gordon Paul Kurtenbach. *The Design and Evaluation of Marking Menus*. Doctor of philosophy, University of Toronto, 1993.

[KWK+10] Dietrich Kammer, Jan Wojdziak, Mandy Keck, Rainer Groh, and Severin Taranko. Towards a formalization of multi-touch gestures. In *ACM International Conference on Interactive Tabletops and Surfaces*, pages 49–58. ACM, 2010.

[LGF10] G.J. Lepinski, Tovi Grossman, and George Fitzmaurice. The design and evaluation of multitouch marking menus. In *Proceedings of the 28th international conference on Human factors in computing systems*, pages 2233–2242. ACM, 2010.

[LHZ+09] Songyang Lao, X. Heng, G. Zhang, Y. Ling, and Peng Wang. 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*, pages 440–446. British Computer Society, 2009.

[Li10] Yang Li. Protractor: a fast and accurate gesture recognizer. In *Proceedings of the 28th international conference on Human factors in computing systems*, pages 2169–2172. ACM, 2010.

[LK99] H.K. Lee and J.H. Kim. An HMM-based threshold model approach for gesture recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(10):961–973, 1999.

[LMZ05] Kang Le Kang Le, Wei Mingxu Wei Mingxu, and Wu Zhongcheng Wu Zhongcheng. An overview of pen computing, 2005.

[McC04] Steve McConnell. *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press, 2nd edition, 2004.

[MH09] Richard Monson-Haefel. What is NUI's WIMP? `http://theclevermonkey.blogspot.com/2009/12/what-is-nuis-wimp.html`, 2009.

[Mic09] Microsoft. *Microsoft Surface - User Experience Guidelines*. 2009.

[Mic12] Microsoft. Windows touch input. `http://msdn.microsoft.com/en-us/library/windows/desktop/dd562197(v=vs.85).aspx`, 2012.

[Nar08] AD Nardi. *Grafiti: Gesture recognition management framework for interactive tabletop interfaces*. Master degree thesis, University of Pisa, 2008.

[ND86] Donald Norman and Stephen Draper. *User Centered System Design: New Perspectives on Human-computer Interaction*. CRC Press, 1986.

[Nie09] Jakob Nielsen. Discount Usability: 20 Years. `http://www.useit.com/alertbox/discount-usability.html`, 2009.

[NN10] Don Norman and Jakob Nielsen. Gestural interfaces: a step backward in usability. *interactions*, (September + October 2010), 2010.

[Nor02] Donald A. Norman. *The Design of Everyday Things*. Basic Books, 2002 editi edition, 2002.

[Nor10] D.A. Norman. Natural user interfaces are not natural. *interactions*, 17(3):6–10, 2010.

[NUI09a] NUI Group. Gesture Recognition. `http://wiki.nuigroup.com/Gesture_Recognition`, 2009.

[NUI09b] NUI Group Authors. *Multi-Touch Technologies*. 1st edition edition, 2009.

[OSK02] Kenji Oka, Yoichi Sato, and Hideki Koike. Real-Time Fingertip Tracking and Gesture. *IEEE Computer Graphics and Applications*, (December):64–71, 2002.

[Pet07] J Petrie. Mixed-fidelity prototyping of user interfaces. *Interactive Systems. Design, Specification, and*, pages 199–212, 2007.

[Pia70] Jean Piaget. *Genetic Epistemology*. Columbia University Press, 1970.

[Ras00] Jef Raskin. *The Humane Interface: New Directions for Designing Interactive Systems*, volume 1 of *ACM Press*. Addison-Wesley Professional, 2000.

[RL04] Eric Raymond and Rob Landley. *The Art of Unix Usability*. 2004.

[Rob99] Jakob Robert. Perl Style Regular Expressions in Prolog. `http://www.cs.sfu.ca/~cameron/Teaching/384/99-3/regexp-plg.html`, 1999.

[RSI96] J. Rudd, K. Stern, and S. Isensee. Low vs. high-fidelity prototyping debate. *interactions*, 3(1):76–85, November 1996.

[RSP11] Yvonne Rogers, Helen Sharp, and Jenny Preece. *Interaction Design: Beyond Human-Computer Interaction (3rd ed)*. Wiley, 2011.

REFERENCES

[Rub91] Dean Rubine. Specifying gestures by example. *ACM SIGGRAPH Computer Graphics*, 25(4):329–337, July 1991.

[Sal10] Laurent Salat. Multitouch and Natural User Interface: Opportunities for a Bottom-Up Approach. *tactineo.com*, 2010.

[SBD+08] Johannes Schöning, Peter Brandl, Florian Daiber, Florian Echtler, Otmar Hilliges, Jonathan Hook, Markus Löchtefeld, Nima Motamedi, Laruence Muller, Patrick Olivier, Tim Roth, and Ulrich von Zadow. Multi-touch surfaces: A technical guide. Technical report, University of Münster, Münster, Germany, 2008.

[SG03] SD Scott and KD Grant. System guidelines for co-located, collaborative work on a tabletop display. *Supported Cooperative Work*, (5), 2003.

[Shn04] By Ben Shneiderman. Designing for Fun : How Can We Design User Interfaces to Be More Fun ? *Computing Systems*, 11(5):48–50, 2004.

[Sny03] Carolyn Snyder. *Paper Prototyping*. Morgan Kaufmann, San Francisco, 2003.

[STG03] Reinhard Sefelin, Manfred Tscheligi, and Verena Giller. Paper Prototyping - What is it good for ? A Comparison of Paper- and Computer-based Low-fidelity Prototyping. *New Horizons*, pages 778–779, 2003.

[Van97] A. Van Dam. Post-WIMP user interfaces. *Communications of the ACM*, 40(2):63–67, 1997.

[Van11] Michaël Vanderheeren. Tabletops and Their Advantages in a Pratical Scenario. *KULeuven*, 2011.

[VRPS11] Norman Villaroman, Dale Rowe, D Ph, and Bret Swan. Teaching Natural User Interaction Using OpenNI and the Microsoft Kinect Sensor. *Human Factors*, 20(9):227–231, 2011.

[VWW10] Craig Villamor, Dan Willis, and Luke Wroblewski. Touch Gesture Reference Guide. `http://static.lukew.com/TouchGestureGuide.pdf`, 2010.

[War09] Todd Warfel. *Prototyping: A Practicioner's Guide*. Rosenfeld, 2009.

[WG10] CG Wimmer and T Grechenig. T. Challenges for Designing the User Experience of Multi-touch interfaces. *Multi-Touch Interfaces*, 2010.

[Wig10] Daniel Wigdor. Architecting next-generation user interfaces. In *Proceedings of the International Conference on Advanced Visual Interfaces*, pages 16–22. ACM, 2010.

[Win96] Terry Winograd. *Bringing Design to Software*. ACM Press, 1996.

[WKZ08] Sabine Webel, Jens Keil, and Michael Zoellner. Multi-touch gestural interaction in X3D using hidden Markov models. In *Proceedings of the 2008 ACM symposium on Virtual reality software and technology*, pages 263–264. ACM, 2008.

[WMW09] Jacob O Wobbrock, Meredith Ringel Morris, and Andrew D Wilson. User-defined gestures for surface computing. In *Proceedings of the 27th international conference on Human factors in computing systems CHI 09*, CHI '09, page 1083. ACM Press, 2009.

[WRFB06] M. Wu, K. Ryall, C. Forlines, and R. Balakrishnan. Gesture Registration, Relaxation, and Reuse for Multi-Point Direct-Touch Surfaces. *First IEEE International Workshop on Horizontal Interactive Human-Computer Systems (TABLETOP '06)*, (Figure 1):185–192, 2006.

[WW11] Daniel Wigdor and Dennis Wixon. *Brave NUI World: Designing Natural User Interfaces for Touch and Gesture*. Morgan Kaufmann, 2011.

[WWL07] J.O. Wobbrock, A.D. Wilson, and Y. Li. Gestures without Libraries , Toolkits or Training : A $1 Recognizer for User Interface Prototypes. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 159–168. ACM, 2007.

[Yan94] J. Yang. Hidden Markov Model for Gesture Recognition. Technical report, DTIC Document, 1994.

[YPBN08] Koji Yatani, Kurt Partridge, Marshall Bern, and M.W. Newman. Escape: a target selection technique using visually-cued gestures. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, number c, pages 285–294. ACM, 2008.

[ZL01] Dengsheng Zhang and Guojun Lu. A comparative study on shape retrieval using Fourier descriptors with different shape signatures. In *Proc. of international conference on intelligent multimedia and distance education (ICIMADE01)*, pages 1–9, 2001.