Touch ‘n’ Sketch: Pen and Fingers on a Multi-Touch Sketch Application for Tablet PC’s

DISSERTAÇÃO DE MESTRADO

Hugo David Jesus Vieira
MESTRADO EM ENGENHARIA INFORMÁTICA

ORIENTAÇÃO
Josef Petrus van Leeuwen
Leonel Domingos Telo Nóbrega

Resumo

Em muitas áreas criativas e técnicas, os profissionais fazem uso de esboços em papel para desenvolver e expressar conceitos e modelos. O papel oferece um ambiente quase livre de restrições, onde eles têm tanta liberdade para se expressar quanto necessitam. No entanto, o papel tem algumas desvantagens, tais como o tamanho fixo e não ter a capacidade de manipular o conteúdo (a não ser removê-lo ou risca-lo), mas que podem ser superadas através da criação de sistemas que podem oferecer a mesma liberdade que o papel, mas nenhuma das desvantagens e limitações. Só nos últimos anos e com o desenvolvimento de ecrãs sensíveis ao toque que também têm a capacidade de interagir com uma caneta, é que a tecnologia tem-se tornado massivamente disponível que permite fazer exactamente isso.

Neste projecto foi criado um protótipo com o objectivo de encontrar um conjunto de interacções mais úteis e utilizáveis, que são compostas de combinações de multi-toque e caneta. Como domínio de aplicação para o projecto foram seleccionadas as ferramentas Computer Aided Software Engineering (CASE), uma vez que abordam uma disciplina sólida e bem definida mas ainda com espaço suficiente para novos desenvolvimentos. Este foi o resultado da pesquisa realizada para encontrar o domínio da aplicação, que envolveu a análise de ferramentas de desenho de várias possíveis áreas e domínios.

Estudos de utilizador foram conduzidos utilizando Model Driven Inquiry (MDI) para ter uma melhor compreensão das actividades humanas e dos conceitos envolvidos na criação de sketches. Em seguida, o protótipo foi implementado, através do qual foi possível executar avaliações de utilizador sobre os conceitos de interacção criados. Os resultados obtidos validaram a maioria das interacções, em face de somente testes limitados serem possíveis no momento. Os utilizadores tiveram mais problemas usando a caneta, no entanto o reconhecimento de escrita e tinta foram muito eficazes e os utilizadores rapidamente aprenderam as manipulações e os gestos da Natural User Interface (NUI).

Palavras-chave

Multi-toque Tablet-PC

HCI

NUI

HAM

Ferramenta CASE de Sketch

Reconhecimento de tinta digital

Abstract

In many creative and technical areas, professionals make use of paper sketches for developing and expressing concepts and models. Paper offers an almost constraint free environment where they have as much freedom to express themselves as they need. However, paper does have some disadvantages, such as size and not being able to manipulate the content (other than remove it or scratch it), which can be overcome by creating systems that can offer the same freedom people have from paper but none of the disadvantages and limitations. Only in recent years has the technology become massively available that allows doing precisely that, with the development in touch-sensitive screens that also have the ability to interact with a stylus.

In this project a prototype was created with the objective of finding a set of the most useful and usable interactions, which are composed of combinations of multi-touch and pen. The project selected Computer Aided Software Engineering (CASE) tools as its application domain, because it addresses a solid and well-defined discipline with still sufficient room for new developments. This was the result from the area research conducted to find an application domain, which involved analyzing sketching tools from several possible areas and domains.

User studies were conducted using Model Driven Inquiry (MDI) to have a better understanding of the human sketch creation activities and concepts devised. Then the prototype was implemented, through which it was possible to execute user evaluations of the interaction concepts created. Results validated most interactions, in the face of limited testing only being possible at the time. Users had more problems using the pen, however handwriting and ink recognition were very effective, and users quickly learned the manipulations and gestures from the Natural User Interface (NUI).

Keywords

Multi-touch Tablet-PC

HCI

NUI

HAM

Sketch CASE Tool

Digital Ink-Recognition

Acknowledgements

Firstly I want to thank my supervisors Dr. Josef Petrus van Leeuwen and Dr. Leonel Domingos Telo Nóbrega for depositing their confidence on me and for accepting to supervise me in this project. Also, I want to thank them for their patience, availability and resilience throughout the bumps in the project.

Secondly I would like to thank all my master's colleagues and friends who always had a helping hand when needed and that demonstrated interested and care for me and this project, with a special thanks to Aquilino, Élvio and Juan.

To the users that participated on the user studies and user evaluations, for their availability when needed and willingness to help and participate, here is a special thanks to them, whom without this project would have not been possible.

Finally, but most important, I would like to especially thank my family which if not for all their support, encouragement and sacrifice I would have not been able to realize my studies and consequently this project.

Index

Figure list

Table list

Acronyms

API	-	Application Programing Interface
ATM	-	Automatic Teller Machine
CAAD	-	Computer Aided Architecture Design
CAD	-	Computer Aided Design
CAP	-	Canonical Abstract Prototype
CASE	-	Computer Aided Software Engineering
CLI	-	Command Line Interface
CMF	-	Compound Multi-Flick
CMOF	-	Complete MOF
CTT	-	Concurrent Task Tree
DFD	-	Data Flow Diagram
DOF	-	Degrees of Freedom
EMR	-	Electro-Magnetic Resonance
EUC	-	Essential Use Case
FTIR	-	Frustrated Total Internal Reflection
GUI	-	Graphical User Interface
HAM	-	Human Activity Modeling
HCI	-	Human-Computer Interaction
ISDOS	-	Information System Design and Optimization System
LCD	-	Liquid Crystal Display
MDD	-	Model Driven Development
MDI	-	Model Driven Inquiry
MFCT	-	Multi-Finger Cursor Techniques
MOF	-	Meta-Object Facility
NUI	-	Natural User Interface
OCGM	-	Objects, Containers, Gestures and Manipulations
PC	-	Personal Computer
PSL/PSA	-	Problem Statement Language/Problem Statement Analyzer
RSVP	-	Rapid Serial Visual Presentation
SDAZ	-	Speed-Dependent Automatic Zooming

SDK	-	Software Development Kit
SILK	-	Sketching Interfaces Like Krazy
UIDST	-	User Interface Design Sketching Tools
UML	-	Unified Modeling Language
WIMP	-	Windows, Icons, Menus and Pointers
WPF	-	Windows Presentation Foundation
XMI	-	XML Metadata Interchange
XML	-	Extensible Markup Language

1 Introduction

Tablet-PCs have been around for quite some time and with them the development of sketch applications that take advantage in using the Stylus to sketch or hand-write. There have been developments in several areas such as architecture, graphic and industrial design, health care, or even more personal uses such as simple note taking and personal journals. Nevertheless one area where there have not been many developments is CASE tools. Some sketching tools exist that allow users to create complete diagrams or even models, but in most cases they are academic and usually not very stable versions.

It is a known fact that when developers and designers are starting a new project they make use of paper sketches for their concepts, as it is an almost constraint free environment where they have as much freedom to express their concepts as they need. However the paper has some disadvantages, such as size and not being able to manipulate the content (other than remove it or scratch it), which can be overcome by creating systems that can offer the same freedom people have from paper but none of the disadvantages and limitations.

1.1 Motivation

With the booming advances in mobile technology there have been recent developments on touch-sensitive screens with dual input digitizers. These developments now open the possibility of interacting with portable devices such as the tablet-pc using a pen as well as fingers.

This provides a new opportunity for more intuitive sketch applications which use a more effective, efficient and more natural combination of pen and fingers input than just finger or just pen interactions. This combination would possibly improve the user's interaction with the tablet.

1.2 Problem and Objectives

As implied in the previews section, the main objective of this project will revolve around sketch applications and combinations of pen and finger interactions.

To find these interactions some sub-goals where established, namely design the interactions, build a prototype to be able to test these interactions, and prove their usability and effectiveness through user testing.

So the main goal in this project "is to find the most useful and usable multi-touch (finger) interactions with pen-enabled interactive screens" (van Lewen, 2010) that are efficient and accurate in an application domain.

Such domain was not initially defined; the general concept of a sketch application was already some idea of what the area could be: CAD tools, digital arts, digital documents, notes, amongst many others in which sketching applications would equally be valid. This yielded the need to research for a suitable project application domain, it then become the first objective of this project, since all other activities would follow in the chosen area.

1.3 Contribution

The expected result is to create a prototype that enables the evaluation of a combination of pen and finger interactions that contribute to improve the initial conceptualization of a sketching CASE tool system.

In the course of this project there will be the need to research and devise the requirements and users' needs for the prototype, and from that a set of guidelines and user needs for sketching CASE tools could be compiled, which could be reused on other projects within the same domain area.

Also a set of natural user interactions for sketching CASE tools in small touch and pen enabled environments, could be devised from the interactions created for the prototype.

1.4 Approach

The approach used was a User Centered Design (UCD) approach.

The fact that initially there was no concrete application domain led to the initial search for that domain. This search for a suitable domain involved the review and analysis of existing sketching tools in general and then a more focused analysis of the tools existing within the candidate domains.

This is a very interaction focused project and with the approach used being UCD, there was participation of users on many steps of the project, mainly on the user studies and the evaluation stages of the development cycles of the prototype.

A user study was conducted using a Model Driven Inquiry (MDI), on the human activities when creating sketches for a new system. The models were created and simultaneously the initial conceptualizations were also created, then the observations and interviews were executed.

An important part of the approach is the cyclic nature of the project development as the results from the user studies influenced heavily on the conceptualizations when they were reviewed afterwards; also as the interactions where conceptualized, integrated into the prototype and then evaluated on user tests, and later adjustments were made to the interactions or new interactions were devised and the cycle would start all over again.

1.5 Document Organization

This document is divided into 7 chapters, and this structure is directly related to the research approach taken to the project.

The first chapter is this one the Introduction where the motivation, problem and objectives, contributions, approach and the structure of the document are stated.

Next the second chapter is the State of the Art, which focuses on describing the context and the current state of the technology in which the project is based upon. It will cover the main concepts and interaction style on the Touch environment, and will take a look on Sketch applications and some related work which will be used also to find the application area

discussed earlier. Then some existing tools namely CASE tools will be analyzed. Also the type of technology that allows this project to exist will be briefly introduced.

Third chapter is Problem Analysis where an inquiry is conducted using the MDI approach mentioned in the previous section and involving the participation of users to have a better understanding of what are the project requirements and user needs.

Then in the fourth chapter is the Concept of the prototype, where the initial concepts, affordances and mapping of the interactions are presented. Also the tasks are described and a storyboard presented.

Fifth chapter relates to the Prototype; and here the application itself will be presented. The hardware and software used will be briefly covered, and then the architecture and the design of the interface as described.

The Validation which is the sixth chapter follows. It is about the user testing and evaluation stages, it will cover all three stages of tests that where performed during the development of the prototype. First are the two initial design-guiding evaluations stages and then the final validation stage.

The seventh chapter, the Conclusion, summarizes what was presented and found throughout the dissertation, some of the difficulties throughout the project. Finally some perspectives on future work; which include future functionalities or extensions that could be implemented.

There are a few appendixes that complement this document. Appendix 1 is about the user studies so it is related to third chapter. Appendix 2 concerns the fourth chapter and Appendix 3 has the material from the evaluations thus is related to the sixth chapter.

2 State of the art

This chapter serves as an overview of the current state of the domain of the project. Since this is a project with a very strong emphasis in HCI this chapter will cover the main concepts and interaction style of the Touch environment. It will be a lot of material but it is all important to provide a strong background and context of the area and what is involved in designing for this new type of interface that is going to be used in this project.

After the initial concepts and aspects of the touch environment, to find the project's application domain, several Sketch applications are analyzed, followed by some related work and some existing tools namely CASE tools as that will be the elected domain.

Finally a historic overview on the technology that allows this project to exist, focusing on the type of technology used in the project.

2.1 The touch environment

This project is based on a recent interface type, the Natural User Interface (NUI). This section presents that interface type and compares it to other interface types. Also mentions some aspects of user computer interaction to which this new paradigm is supportive. And then some guidelines that helped shape the design of the prototype.

2.1.1 Interface type NUI

To better understand what a natural user interface is, here is an analysis of the role and context of the NUI. As it is "the next generation of interfaces" and is possible to interact with these interfaces "using many different input modalities, including multi-touch, motion tracking, voice, and stylus", our input options are increased. However "NUI is a new way of thinking about how we interact with computing devices and it is not just about the input" (Blake, Natual User Interfaces in .NET, 2011).

The definition of the Natural User Interface as Blake (Blake, Natual User Interfaces in .NET, 2011) defines it:

> *"A natural user interface is a user interface designed to reuse existing skills for interacting directly with content".*

From this definition we can point out a few important aspects:

- It says that the interfaces are designed which means that they are planned to have appropriate interaction for the user and the content.
- Because users are humans, they possess skills that where gained through their lives. NUIs reuse those skills and make use of today's technology to give users more natural interfaces.
- The content can be directly interacted with; this means that manipulating content directly should be the primary method of interaction, however the interface can have controls such as buttons when necessary, but they should be secondary to content direct interaction.

Bill Buxton (Buxton, "CES 2010: NUI with Bill Buxton" at MSDN's Channel 9, 2010) has another definition of NUI:

> *An interface is natural if it "exploits skills that we have acquired through a lifetime of living in the world."*

It is interesting here that not only is mentioned that the skills are reused so the user does not have to necessarily learn new skills, but it also clearly states these abilities are not only one's inborn abilities but the skills that are learned throughout life.

Further ahead in the chapter these abilities and their aspects will be explored in more detail.

2.1.1.1 *Comparing NUI vs. GUI and CLI*

Comparing NUI to other interface types will make it easier to understand the definitions presented above.

Graphical User Interfaces (GUIs) uses Windows, Icons, Menus and Pointers (WIMP); where windows, menus, and icons are "artificial interface elements" for output and a pointing device (such as a mouse or a touchpad) for input. The Command Line Interface (CLI) has "text output and text input using a keyboard" (Blake, Natual User Interfaces in .NET, 2011). NUIs use the Objects, Containers, Gestures and Manipulations (OCGM) interaction style; which roughly means that objects and containers are used for output, gestures and manipulations are used as input, in the next section we will see OCGM in more detail.

The prime difference between these interface types is not only in the input modality they use (keyboard, mouse or touch). Also "CLI and GUI are defined explicitly in terms of the input device, while NUI is defined in terms of the interaction style". Whereas NUI can use any type of device or interface technology, which interaction style if focused on reusing existing skills (Blake, Natual User Interfaces in .NET, 2011).

GUI "uses windows, menus, and icons for primary interface elements" (Blake, Natual User Interfaces in .NET, 2011), contrasting with that, NUI interacts directly with content. So one would assume that it does not need controls, however, as said in the previous section that assumption would not be correct. This fact only means direct manipulation of the content should be the primary interaction method and controls should be secondary to the content.

Blake (Blake, Natual User Interfaces in .NET, 2011) says "GUI is the stale technology and NUI is the more capable, easier to learn, and easier to use technology". So NUI will not take the place of exiting GUIs or even CLIs, "the world migrated from CLI application to GUI applications because GUI was more capable, easier to learn, and easier to use in everyday tasks". Just as happened with CLIs, NUIs will be used on the more general purpose interfaces; however GUIs will still be used for those specialized tasks, where despite being limited to mouse and keyboard, it is the most effective way to accomplish them.

For instance might be the case of a "mixed" user interface where the traditional pointing device may be necessary for precise pointing tasks, and yet the interface would be considered as NUI even if it still uses components of GUI. Since NUI is not about input devices, but about

interaction style, “it would be valid to design a NUI that used keyboard and mouse as long as the interactions are natural” (Blake, Natual User Interfaces in .NET, 2011).

2.1.2 OCGM interaction style

As said in the first chapter NUI uses OCGM as the interaction style, this section explores its components.

Ron George (George, "Welcome to the OCGM Generation! Part 2" at Ron George Blog, 2009), defines this as “OCGM breaks down the basis of all future interfaces into two categories, (…) one for Items and one for Actions. (…) Those are broken down into two subcategories” each; Objects and Containers for the Items, and Manipulations and gestures for the Actions.

Objects

Objects can be anything and take any shape on the interface, a picture, a textbox, an icon or a button. An object basically represents something or an action of the system. Having objects being so coverable, helps to free developers of thinking in Icons and Windows, and explore other ideas.

Containers

A Container represents the bonds between objects. In an interface, it takes shape as the relationship existing amongst objects, and it is not confined to the traditional form of a physical box or window. For instance “they could be 5 balls circled around a larger ball which forms a sort of a menu. They could be a simple tagging system”, that when activated could “reveal the tagged objects and therefore reveal the container” (George, "Welcome to the OCGM Generation! Part 2" at Ron George Blog, 2009).

One can consider manipulations and gestures as being objects, and in this sense they could also be enveloped by containers. Furthermore Containers could be considered as objects themselves. Therefore the interface is composed of objects and the relationships between them, which are the key to manage the objects, and understanding these relationships is a fundamental part of the design.

Manipulations and gestures

The main difference between Manipulations and Gestures is the nature of the interaction, as represented in Figure 1.

Manipulations are the natural direct interactions mentioned before. They are easy to perform, to understand and fairly intuitive. They are more appropriate to beginners and medium users, and should be designed to be accidentally activated, thus they should be easily discovered. A manipulation usually has an immediate reaction associated on the interface; this gives feedback to the user making it easy for him to understand the result of his action.

Gestures are the opposite of Manipulations. They are indirect complex actions that “are usually not intuitive (draw a ? for help), and are not geared towards the first user experience” (George, "Welcome to the OCGM Generation! Part 2" at Ron George Blog, 2009).

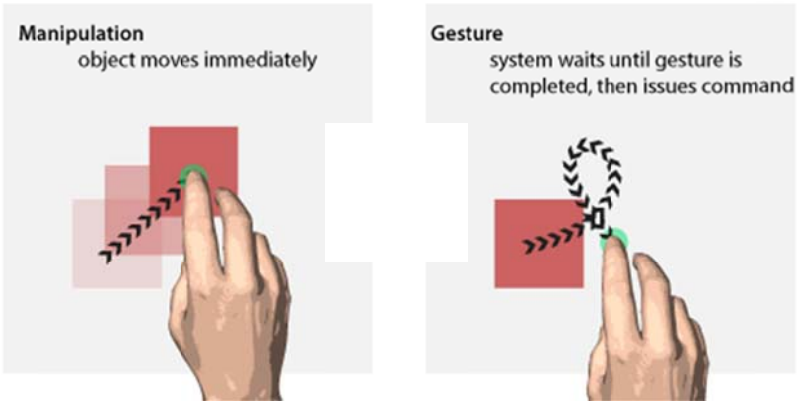


Figure 1: Gestures vs. Manipulations. (George, "Terminology: the difference between a gesture and a manipulation" at Ron George Blog, 2009)

Overall the interface should be designed for accidental activations and they should always be geared towards a Manipulation and not to a Gesture, as Manipulations can be used to perform most of the common daily tasks that the user needs (George, "Welcome to the OCGM Generation! Part 2" at Ron George Blog, 2009). For example swiping the hand to the left on a touch screen, which is something that could happen by accident very frequently, should never be assigned to a harmful or unrecoverable action such as delete a file or clear the screen.

Another interesting example is “to start the self-destruct on a ship,” one does not simply press a button. Usually one has to “perform a gesture, several manipulations in a sequence that are recognized at the end of the sequence. Only then, after the order is maintained and accomplished does a gesture get recognized and then the action is performed.” (George, "Welcome to the OCGM Generation! Part 2" at Ron George Blog, 2009)

Ron describes four differences between Manipulations and Gestures that makes it easier to distinguish them:

Table 1: Differences between Manipulations and Gestures. (George, "Terminology: the difference between a gesture and a manipulation" at Ron George Blog, 2009)

Manipulations	Gestures
Contextual – they only happen at specific location(s) or on specific object(s)	Not contextual – they can be anywhere in the system in location and time
React immediately – there is a direct correlation in cause and effect between your interaction and the system (this does not include visual affordance)	The system waits for the series of events to complete to decide on how to react (again, this does not include visual affordance)
Can be single state, but are usually 3 or more states (see Bill Buxton’s paper on Chunking and Phrasing (Buxton, Chunking and Phrasing and the Design of Human-Computer Dialogues, 1986))	They contain at least 2 states
Direct (could possibly be considered indirect by way of augmenting your actual interactions with the reaction of the system) – your actions directly affect the system, object, or experience in some way	Indirect – they do not affect the system directly according to your action. Your action is symbolic in some way that issues a command, statement, or state.

2.1.3 Using the users: Innate abilities and learned skills

As mentioned above natural interfaces try to reuse skills and abilities the user possesses already, in a way in NUI the users are used to improve the design and the usability. In this section an overview of what are those skills and abilities is presented.

Humans "are all born with certain abilities", for instance "the ability to detect changes in our field of vision, perceive differences in textures and depth cues, and filter a noisy room to focus on one voice" (Blake, Natual User Interfaces in .NET, 2011). Other abilities develop as one matures, and examples of those are the abilities to eat, walk or talk.

The ability to learn

Humans also have a very important innate ability which is the ability to learn. However despite skills and abilities being used to accomplish tasks, "learned skills are different than innate abilities because we must choose to learn a skill, whereas abilities mature automatically" (Blake, Natual User Interfaces in .NET, 2011).

Using this ability to learn humans build upon the simple innate abilities that we use to learn skills and to accomplish simple tasks. By building these skills based on existing skills we gradually learn more complex skills and can increasingly perform more complex tasks. Then one can subdivide skills into two categories: simple skills and composite skills.

Simple skills

"Simple skills are learned skills that only depend upon innate abilities. This limits the complexity of these skills, which also means simple skills are easy to learn, have a low cognitive load, and can be reused and adapted for many tasks without much effort" (Blake, Natual User Interfaces in .NET, 2011).

An example is tapping, because it is a natural human behavior which to master only requires the innate ability of fine eye-hand coordination (Blake, Natual User Interfaces in .NET, 2011). Tapping can be easily reused in interfaces, it could for instance be used to select an element from the interface or to activate a button, just like it is used in everyday situations to call attention to an object or to push a button on the radio of a car.

Composite skills

"Composite skills are learned skills that depend upon other composites or skills, which means they can enable you to perform complex, advanced tasks. It also means that relative to simple skills, composite skills take more effort to learn, have a higher cognitive load, and are specialized for a few tasks or a single task with limited reuse or adaptability". (Blake, Natual User Interfaces in .NET, 2011)

Going back to the previous example, tapping is a simple skill but clicking with the mouse is not, they often accomplish the same result but the actions are different. The mouse click "is a composite skill because it depends upon the skills of holding and moving a mouse and acquiring a target with a mouse pointer. Using those two skills together requires a conceptual understanding of the pointing device metaphor". (Blake, Natual User Interfaces in .NET, 2011)

Blake (Blake, Natual User Interfaces in .NET, 2011) indicates three reasons as to why it is a composite skill:

- **More effort to learn** - We must invest a lot of practice time with the mouse before we can use it quickly and accurately.
- **Higher cognitive load** - Mouse skills fall towards the basic side of the skill continuum, but the mouse still demands a measurable amount of attention.
- **Specialized with limited reuse** - the master mouser skills you spent so much time developing have no other applications besides cursor-based interfaces.

For experienced users, using the mouse is easy as they do not even think about it, but it is not so straight forward for less experienced users. Practice and conscious effort are required to accomplish a significant level of skill with the mouse or with any other composite skill.

Using skills increases cognitive load

"Cognitive load is the measure of the working memory used while performing a task. The concept reflects the fact that our fixed working memory capacity limits how many things we can do at the same time." (Blake, Natual User Interfaces in .NET, 2011). John Sweller (Sweller, 1988) developed this theory on cognitive load.

"Cognitive load theory states that extraneous load should be minimized to leave plenty of working memory for germane load, which is how people learn. The inherent difficulty of a subject cannot be changed, but the current intrinsic load can be managed by splitting complex subjects into sub-areas." (Blake, Natual User Interfaces in .NET, 2011). This can be applied to many areas but to interface design this means that the cognitive load created by the skills used in the interaction should be minimal, it is more important to leave sufficient space in the working memory for the load involved in learning the interface.

In relation to skills this means that the interface should be designed to make use of simple skills rather than composite skills, this does not mean that the later cannot be used when appropriate.

2.1.4 NUI Guidelines

As in other interface types and styles, there are some rules of thumb, patterns or templates to guide designers through their process, and NUI is no exception. Blake (Blake, Natual User Interfaces in .NET, 2011) suggests four guidelines based on the abilities, skills and cognitive loads above.

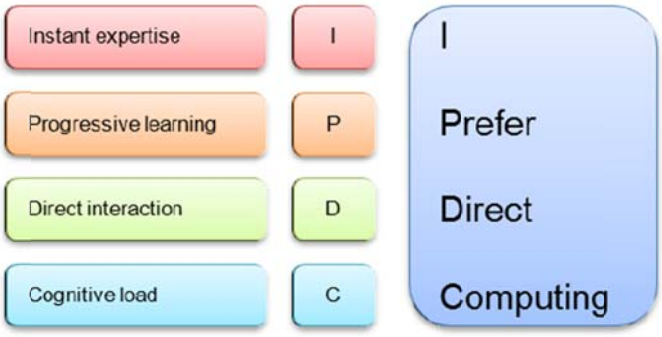


Figure 2: Blake's motto for NUI guidelines. (Blake, Natual User Interfaces in .NET, 2011)

2.1.4.1 *Instant expertise*

This guideline says that designers should reuse the skills users already possess when they are designing interactions. In this way users will not have to learn new skills to use the interface, they will be instant experts as they do not have to spend much effort or time getting up to speed.

Instant experts can be created by one of two ways (Blake, Natual User Interfaces in .NET, 2011):

- Reusing domain specific skills – Targeted users already have most of those skills, but there can be differences in the skills within the target group population. Also most domain specific skills are not easy to use in new situations, because they are composite skills.
- Reusing common human skills – Since humans will be the users, reusing simple skills that they have developed just by being human is a better approach with most scenarios.

2.1.4.2 *Cognitive load*

The cognitive load guideline states that one "should design the most common interactions to use innate abilities and simple skills", because "the majority of the interface will have a low cognitive load and be very easy to use", and "the interface will be very quick to learn, even if some or all of the skills are completely new" (Blake, Natual User Interfaces in .NET, 2011).

This guideline could conflict with the instant expertise guideline when users possess a composite skill already. For instance, in the case of a user that has the composite skill of using the mouse, an interface where most of the interactions require the user to learn new simple skills, still has a lower cognitive load than the same interface where the interactions are mostly performed with the mouse. This is because using the mouse is a composite skill and has a higher cognitive load than innate abilities or simple skills. And the cognitive load of learning the new simple skills will no longer weight the balance when the user has learned the skills. In the case of not being possible to reuse simple skills, teaching simple skills should be the priority and not the reuse of composite skills, as over time "teaching and using new simple skills requires less effort and is more natural" (Blake, Natual User Interfaces in .NET, 2011).

2.1.4.3 *Progressive learning*

In general HCI design, "a smooth learning path from basic tasks to advanced tasks" should be provided to the user. As NUIs are no different, they should "enable the user to progressively learn from novice to expert" but "at the same time, the interface should not get in the way of expert users doing advanced tasks". (Blake, Natual User Interfaces in .NET, 2011)

Usually advanced tasks get broken down into simpler subtasks that are easily handled. In an interface simpler tasks usually require simple skills and advanced tasks require complex skills. When is not possible to subdivide an important advanced task into simpler tasks, that means that a complex skill has to be used to achieve that task. In this case the task should not be a part of the core interface for beginner users, and complex tasks should be limited, used only for the crucial tasks that cannot be done in any another way.

2.1.4.4 *Direct interaction*

This guideline states the interface should be designed "to use interactions that are direct, high-frequency, and appropriate to the context" (Blake, Natual User Interfaces in .NET, 2011). To understand this guideline one needs to know what are direct, high-frequency and contextual interactions.

Direct Interactions

There are three types of direct interactions, as illustrated on Figure 3. The top of the figure illustrates Spatial Proximity; the finger is touching over the object itself. In the center of the figure there is Temporal Proximity. The user is acting and the interface is reacting at the same time (real time). At the bottom is the third type of directness Parallel Action, where the reaction of the interface is parallel to the action of the user in at least one degree-of-freedom.

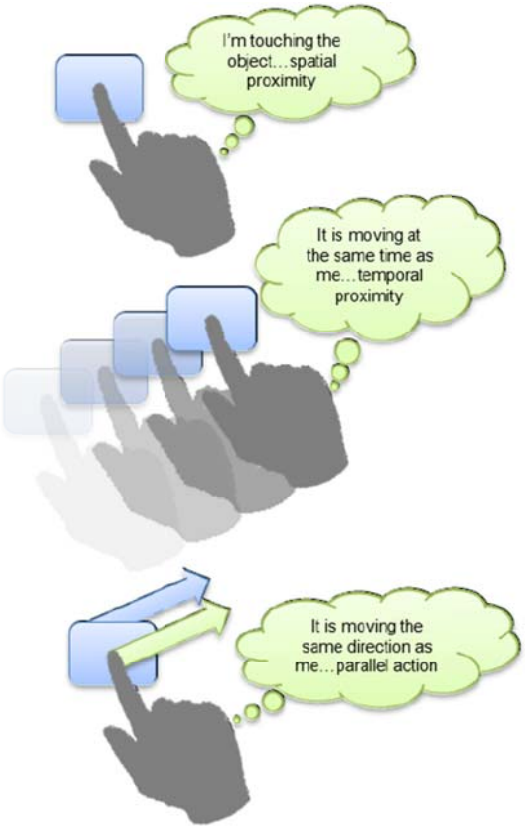


Figure 3: Types of Directness illustration. (Blake, Natual User Interfaces in .NET, 2011)

In the real world one interacts directly with objects, which in the interaction world is the content, and we have a natural sensory-rich experience doing it. For instance when someone drags a sheet of paper over a table, in a brainstorm session for example, it follows your hand movement at the same time and in the same direction, so it is a direct interaction which reaches the three types of directness. As one drags the sheet over the table, concentrating only on the senses of the hand, one feels on the fingertips the texture of the paper, the pressure of the exerted force and the resistance of the drag, and as the fingers and hand bend one can instantly perceive in which direction the sheet is moving. Without the visual feedback there are already multiple cues that are enough to let the person know how and what they are manipulating.

Humans naturally use hands and fingers to directly manipulate real objects and become better at it as they grow and develop. Interactions which make use of the simple skills and innate abilities humans already own, and "that are direct also tend to be faster than interacting indirectly with content through a mouse and interface elements" (Blake, Natual User Interfaces in .NET, 2011). Also they tend to be smaller thus they can be high-frequency interactions.

High-frequency interaction

"Interfaces that allow many quick interactions provide a more engaging and realistic experience. Each individual interaction may have only a small effect, but the bite-sized (or finger-sized, perhaps) interactions are much easier and quicker to perform. This allows the

user to perform many more interactions and get much more feedback in the process than a GUI-style interface would allow" (Blake, Natual User Interfaces in .NET, 2011).

When users are for instance near or crossing a limit of the interaction they are performing, the more frequent feedback from quick interactions helps users to have an easier and faster comprehension of it. This is because they are faster to process than less frequent bigger feedback, which is usually what one gets from the more significant actions in GUIs. High-frequency can also help users to know the reasons of not being able or having the ability to do something in the interface.

Contextual interactions

"The context of an interaction includes what action the user is performing, the proximity of the action to certain visuals, and the previous interactions before the current one. In order to create contextual interactions and have the feedback interpreted properly, it is important to reduce the number of possible tasks to only those that are appropriate at the time. Fewer tasks also means fewer interface elements shown at once. A side-effect of this is that interfaces have minimal clutter." (Blake, Natual User Interfaces in .NET, 2011).

When humans are presented with many possibilities or options they can become overloaded and not able to choose one, this is called choice overload, and in the same way this can happen with the available tasks on an interface. Although having all the possible tasks available at once allows the user to quickly transition to any activity with minimal interaction, which may seem powerful as it reduces the amount of effort to switch activities, it can result in choice overload.

Choice overload can be controlled in an interface by only presenting the essential tasks for the activity that the user is performing. This does not mean that functionalities have to be removed from the interface, only that the ones which are not needed for any given activity do not have to be present in the interface within the context of the activity. This way the interface becomes simpler and cleaner, and the user can make faster and easier choices.

2.2 Related Work and Domain Research

The applications for tablet-pc technology are very diverse and go from the simplest personal uses to the more serious industrial purposes. This created an issue on the project which was what would be the application domain of the project. So for purposes of defining that domain, a research on the areas of tablet-pc sketch applications was conducted.

A few areas were selected to find a research opportunity for touch and pen interaction. The research involved the analysis of the results of the development of several sketching tools and techniques.

In this section is presented an overview of that research and the conclusions and factors that resulted on the focus application domain for this project.

2.2.1 CAD, CASE and Sketch tools

These domains are reviewed together because the cases researched in this project provide support to free-hand sketching as well as recognition of those sketches. In other words

the tools reviewed here are not the traditional widget based CAD or CASE tools, they are sketch based.

2.2.1.1 *The Sketching Paradigm*

"Designers initially hand-sketch their ideas because informal tools, such as pen and paper, offer the freedom to work with partly formed or ambiguous designs" (Chung, Mirica, & Plimmer, 2005).

So "for early conceptual and creative stages of design", users need a "natural and unobtrusive" interface "in order to avoid interfering with the designer's creative processes" and also the "user needs flexibility in the interface to accept whatever is done without fuss to foster maximum creativity and capture the essence of a design" (Dickinson, Yu, Zeng, & Antunes, 2005).

Sketching in Architecture CAD tools

Regarding to CAAD tools (such as EsQUIsE prototype (Juchmes, Leclercq, & Azar, 2006)), results show that sketching application and Tablet PC environment are suitable for architects' needs, also "the ability to simultaneously sketch on various areas enables architects to keep their own style of designing" and they "do not restrain themselves to a narrow exploration of the solution space". And "in terms of interaction process (…) and graphical production (…), the tablet seems suitably supporting architectural design activities" (Darses, Mayeur, Elsen, & Leclercq, 2008).

3D in CAAD is rated highly, but when sketching architects do not produce 3D sketches in a large volume. Possible reasons are that the use of 3D (either drawn or CAAD generated) at the initial phase of the design process, sketching, there is no spatial reasoning and that the use of 3D varies widely among architects. So even though 3D support is very important in this area, it seems that for the initial stages the real need is for 2D sketch support and recognition (Darses, Mayeur, Elsen, & Leclercq, 2008).

Sketching in 3D

Teddy (Igarashi, Matsuoka, & Tanaka, 1999) is a sketching application that has a simple interface which allows for rapid sketching of 3D shapes based on freedom strokes drawn in a 2D perspective.

It creates objects based on their 2D stroked silhouette, and then the user can perform modification operations on the 3D model, such as surface painting, erasing, extrusion, cutting, carving, smoothing, and transformation operations like bending and distorting. To perform these operations Teddy allows for direct camera manipulation, so it is possible to see the shape from different perspectives (Igarashi, Matsuoka, & Tanaka, 1999).

The program has a very freehand experience and its gesture based manipulation seems intuitive to use, however in the studies made, users had a 10 min tutorial and guided practice. Regardless the program is a simple and effective way of rapid prototyping ideas and concepts in a simple handcrafted look 3D model (Igarashi, Matsuoka, & Tanaka, 1999).

2.2.1.2 *Recognizing the Sketch*

When sketching "users never announce the action they intend to perform (no selection in the menus)" (Juchmes, Leclercq, & Azar, 2006), they just draw. In this environment the application must be able to understand what the user is doing. So several technics are used to find want what the user wants to do.

2D Extrapolation

In Teddy (Igarashi, Matsuoka, & Tanaka, 1999) the sketches are recognized by applying algorithms to the user's 2D sketches, extrapolating them into 3D shapes. The 2D silhouettes are inflated into 3D models, like they were balloons. These algorithms work well for a variety of manipulations, but sometimes create unexpected results when the user draws unusual strokes. And because of this way of interpreting the base sketch Teddy has its limitations, like the shapes produced, because they have to be spherical based; the user cannot create a torus for example.

Agents

One solution that is very effective in recognizing users' stokes, is a multi-agent system as the one implemented in EsQUIsE (Juchmes, Leclercq, & Azar, 2006).

In this solution the agents are "able to recognize the user's intention and act accordingly" as the EsQUIsE (Juchmes, Leclercq, & Azar, 2006) interface is very simple to allow as much freedom to the user as he or she would get form a paper based sketch. For that purpose the agents are divided in multiple levels, which help in processing different kinds of data. The lower level and therefore the more simple agents act immediately when an input is detected, and if that input is recognized as belonging to the agent it quickly stops further processing from other agents and adds the input to the model. The higher level agents are assigned to more complex tasks and even do not work directly on the strokes drawn by the user, but use a result produced by other agents as input. For example this is the case with caption recognition.

These agents are made to collaborate amongst themselves and this "collaboration between two agents makes it possible to obtain a more pertinent interpretation" (Juchmes, Leclercq, & Azar, 2006) of complex inputs as they may combine several simple stokes or elements into a complex shape, and without this dialogue between agents they would have been interpreted as separate things instead, or might have not been interpreted at all.

Another point in favor of multi-agent architecture vs. single agent is that the feedback to the user can be made available more quickly even in more complex sketches and also using a modular architecture (having several agents instead of one), makes it easier to add or remove elements as necessary, since it is done "with the least possible repercussions on the system" (Juchmes, Leclercq, & Azar, 2006).

So this multi-agent approach is very well adapted to this sketch recognition as "the autonomy of the agents makes it possible to handle the free expression characterizing the sketch being drawn, while the collaboration among the agents enables them to carry out complex tasks" (Juchmes, Leclercq, & Azar, 2006).

The Toolkit Approach

Generally the existing sketch tools support only specific design tasks because the effort required to build a wider supportive computer aided sketching tool is too great. So in this panorama a toolkit approach to sketch recognition was suggested in InkKit by Plimmer and Freeman (Plimmer & Freeman, 2007) in an effort to cut on the development effort to support a specific domain.

"In a computer-supported environment (…) recognition is essential for more sophisticated support, it facilitates automatic conversion of sketches to formal diagrams and the automation of sketches". "A natural sketching environment consists of both words and characters; the recognition engine must manage" these two different inputs. "Some of the areas under investigation are automatic conversion from sketch to formal diagram, and supporting automation; this requires a recognized sketch and knowledge of the underlying semantics" (Plimmer & Freeman, 2007).

A general approach to recognition such as implemented in InkKit is possible. Without regard for domain, ink is divided into writing or drawing strokes, this is called modeless recognition. "Writing strokes are recognized by a text recognizer. Drawing strokes are joined to form basic shapes that are recognized by" an enhanced version of Rubine's algorithm, developed by Plimmer and Freeman (Plimmer & Freeman, 2007), "thus overcoming the main weakness of this algorithm".

"Domain specific information is required to compose components from the words and basic shapes". In InkKit this is done "from user examples, extracting from these examples the basic elements and their spatial relationships. Further more specific knowledge is required to convert a sketch to a formal representation or automate it. InkKit makes available the recognition information via an API so that these next stages are easy to program" (Plimmer & Freeman, 2007).

The InkKit implementation "provides context free design spaces and a powerful, trainable and extensible modeless writing/drawing recognition engine" (Plimmer & Freeman, 2007). This possibility of easily extending the scope of domains that a sketch tool can support is what makes this toolkit approach attractive.

2.2.1.3 *CASE tools*

As stated before, "flexibility is particularly important in the early stages of UI design itself, when designers need the freedom to sketch rough design ideas quickly, the ability to test designs by interacting with them, and the flexibility to fill in the design details as they make choices" (Landay & Myers, 2001). One tool that intended to provide designers with this flexibility was SILK (Sketching Interfaces Like Krazy) (Landay & Myers, 2001). It is an effective informal design tool for a specific creative task, sketching interfaces.

What SILK can do to help?

"Designers can quickly sketch an interface using an electronic pad and stylus, and SILK recognizes widgets and other interface elements" in real time (as the designer draws them). Moreover "designers can test the interface at any point, not just when they finish the design";

they can exercise the widgets that they drew. “For example, a sketched scrollbar is likely to contain an elevator or thumbnail, the small rectangle a user drags with a mouse”. A SILK sketch supports designers dragging it up and down, “which lets them test the components’ or widgets’ behavior. SILK also supports the creation of storyboards”. And after the sketch is done designers can tell SILK to “transform the sketch into an operational interface using real widgets, according to a particular look and feel” (Landay & Myers, 2001).

Remarks

The usability test preformed to the tool “showed that electronic sketching effectively supports the early stages of UI design”, despite the problems that revealed by it, “particularly widget recognition”. This problem will require more research, but since recognition is done by computing the spatial relationships between the primitive components, SILK could learn a widget using that ability (Landay & Myers, 2001).

“But even with its current recognition accuracy (…) designers were able to move quickly through several iterations of a design using gestures to edit and redraw portions of the sketch”, that in this tool is recognized by Rubine’s algorithm. And they also “found SILK to be effective for both early creative design and for communicating the resulting design ideas to others” (Landay & Myers, 2001).

2.2.1.4 *Conclusion*

Sketching CAD and CASE applications it is an area where very much can still be explored, and enough has been done that can serve as a base for this project. Namely the recognition of sketches is still an area in progress, but for simple to medium complex diagrams provides enough support that can be used for further research on other aspects like the interaction with the user.

In this area there has to be big concern with the sketch itself, recognizing it and producing useful and reusable information from it. Not just keeping it as an initial drawing of the system and not getting something else out of it.

Relating to sketch driven design in CAD modeling, “Pen-Tablet has demonstrated significant potential, mostly because many existing mouse interaction paradigms can be transferred to pen-based application, but also, some interaction paradigms that users are familiar with from simple pen/pencil and paper activities commuted to digital applications” (Dickinson, Yu, Zeng, & Antunes, 2005). The same can be said to CASE tools like SILK (Landay & Myers, 2001) that use sketching as input, and that users where accustomed to do those sketches with paper and pencil.

2.2.2 Digital Documents

In this area the research concentrated on the navigation in digital documents and not on edition. And in this perspective several navigation techniques were investigated, bellow are these techniques and results that arose from their study.

Radial Scroll

One technique was the radial scroll, which is “an interface widget to support scrolling”. With this widget “users gesture clockwise anywhere on the document surface such that gestures advance the document and counter clockwise gestures reverse the document”. “For short scroll target acquisition tasks, to which continuous scrolling is suited, radial scroll performs better than traditional techniques”. For long distance scrolling, speed still needs to be improved (Smith, 2004).

This tool still needs improvement but it seems to have potential to be a better way than the traditional scrollbar for scrolling particularly on small and large touch-based devices (Smith, 2004). This because on small screens the user no longer has to point and drag the thin elevator of the scroll bar, and for the large devices the user can navigate through a document despite his ability to reach the scroll bar from his current position.

Flipper

Another technique is the Flipper (Sun & Guimbretière, 2005), a “digital document navigation technique inspired by paper document flipping. Flipper combines speed-dependent automatic zooming (SDAZ) and rapid serial visual presentation (RSVP)”. “Flipper effectively carries the affordance of page flipping to the digital world” and “it not only provides users with a way to perform a rapid visual search, it also offers a smooth transition between fast and slow document navigation”.

As shown in the Figure 4, “as speed (shown on the graph) increases, Flipper switches from scrolling to page flipping. An abrupt change in speed at 4.5s causes an automatic backtrack. Each image illustrates the state of the screen at one second interval. Page labels were added for clarity” (Sun & Guimbretière, 2005).

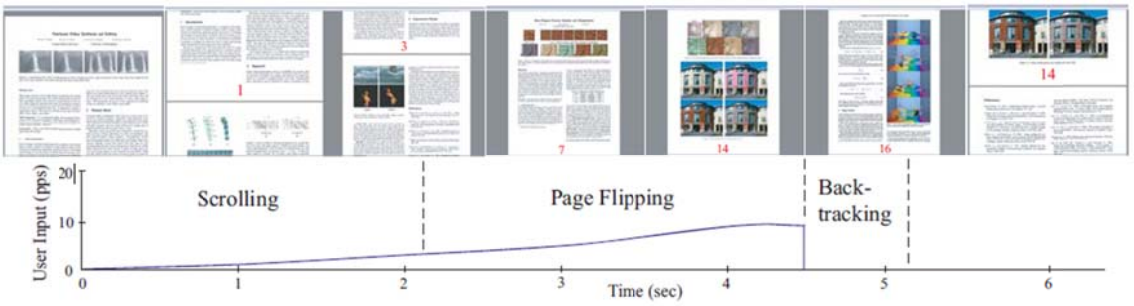


Figure 4: Flipper's Behaviour. (Sun & Guimbretière, 2005)

Tests “show Flipper is the fastest method in testing environment (Flipper, SDAZ and a simulation of the Acrobat Reader’s thumbnail system) and that its advantage increases with greater document size”. “User studies show Flipper is faster than both conventional scrolling and SDAZ and is well received by users” (Sun & Guimbretière, 2005).

Despite this technique was not developed for pen or touch based devices it can be adapted, especially to pen environments, as it was mentioned above that many existing mouse interaction paradigms can be transferred to pen-based application (Dickinson, Yu, Zeng, & Antunes, 2005).

Multi-Flick

The last technique is Multi-flick (Aliakseyeu, Irani, Lucero, & Subramanian, 2008), "which consists of repeated flick actions" to the sides with a stylus as in Figure 5, it "has received media attention as an intuitive and natural document-scrolling technique for stylus based systems". A comparison of three flicking techniques and the traditional scrollbar shows that of these three techniques, "compound multi-flick (CMF) is the most preferred technique" and that it is at least as fast the traditional scrollbar. CMF consists of a combination of techniques, as the name compound indicates, but the improvement here is that it provides the user with feedback, displacing the document in "a distance equivalent to the displacement of the pen", as they are executing the flicking and not just after the pen is lifted, as happened with the other two.

Figure 5: Multi-flick scrolling technique. (Aliakseyeu, Irani, Lucero, & Subramanian, 2008)

"Results show that all implementations of multi-flick are as good as the scrollbar for short distances while CMF is the most preferred" and the closest to the speed of the scrollbar when long distance scrolling is performed. The CMF is preferred to the conventional scrolling because, besides being a more natural form of interaction, it is less fatiguing to the user (Aliakseyeu, Irani, Lucero, & Subramanian, 2008).

2.2.2.1 *Conclusion*

In digital documents navigation, the techniques required are the common and well established navigation techniques, which are fairly dispersed by touch and multi-touch devices, mobile and otherwise. So there is not too much space for multi-touch pen and fingers innovation.

2.2.3 Digital Arts

"Physical experience and materiality in creative digital visual art and design practice has a special significance". Treadeway (Treadaway, 2009) indicates that "memory of physical engagement with the world is crucial for creative cognition", in other words, "making by hand, touch and manipulative activities, have a significant impact on creative thinking and imagination".

In this perspective the feedback provided to the users by the interfaces of digital tools and input systems is how the users obtain that memory of physical engagement. However the current tangible interfaces available that provide tactile information to users are economically unviable for the most and "remain relatively inefficient at conveying the rich sensory experience of the hands and fingers". So the development of more tangible interfaces, namely in "sensing technology and haptic interfaces will provide greater potential to translate physical experience into digital format" (Treadaway, 2009).

2.2.3.1 *Conclusion*

Since the "production of usable commercially available haptic devices lags well behind academic research" (Treadaway, 2009), this area does not seem ideal for this project. That is because the feedback and sensory experience that current Tablet PC systems can provide does

not fit in the one required by the digital arts area and the scope of this project does not fall outside the Tablet PC technology.

2.2.4 Touch and Multi-Touch

During the initial research some general touch and multi-touch aspects and techniques have been analyzed. And to study and establish those principles, techniques and aspects of multi-finger interaction, researchers often develop applications and interfaces that allow them to apply and test their theories and hypothesis. Here are two of those applications.

2.2.4.1 *Multi-touch interaction*

To the end of establishing principles of multi-finger interaction Moscovich (Moscovich, 2006) developed two multi-finger interfaces to test several interaction techniques.

Multi-Touch Performance Animation

By using a projector pointed at a multi-point touchpad and a novel deformation algorithm, Moscovich (Moscovich, 2006) was able to produce an interface that "allows users to move and bend drawings by touching them with their fingers as though they were physical rubber props". The movements are spread through time at a real time rate and captured, thus producing an animation. So the function of this application was to create 2D animations.

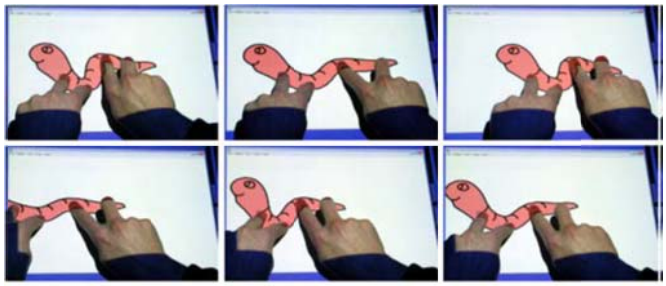


Figure 6: "User bends a drawing by touching it with his fingers". (Moscovich, 2006)

"Traditional key-framing techniques tend to produce stilted motions in the hands of novices", compared to that this "technique yields more believable results, since it preserves all of the nuances and imperfections of the user's hand motion" (Moscovich, 2006).

Multi-Finger Cursor Techniques

With another interface which was a drawing program, Moscovich (Moscovich, 2006) tested several Multi-Finger Cursor Techniques (MFCT). "An important feature of these methods is that hand parameters are not mapped directly to the manipulated object. Instead, they control an intermediary representation, the cursor" through a multi-touch pad, which "is used to simultaneously select the object to control and then transfer its properties to it".

The first MFCT that was tested (shown in Figure 7 left) "shows a hand cursor, which displays the finger touch-points (…) relative to the hand position on the screen. To the user, moving objects on the screen feels like sliding flat objects on a table. Users can select multiple objects using individual fingers, and can simultaneously translate and rotate objects. Users can also control objects with many degrees of freedom, for example (…) the control points of a curve" (Moscovich, 2006).

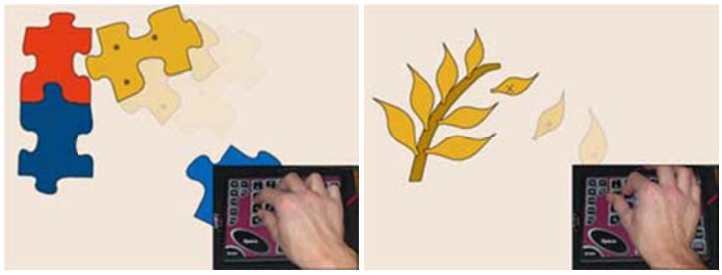


Figure 7: Multi-finger cursor techniques: multiple cursors (left) and single cursor (right). (Moscovich, 2006)

"Since users generally control only one object at a time, it is useful to abstract the parameters of the hand into a single point cursor". So the second MFCT (in Figure 7 right) "allows the user to focus on a single target, while simultaneously controlling its position, rotation, and scale" (Moscovich, 2006).

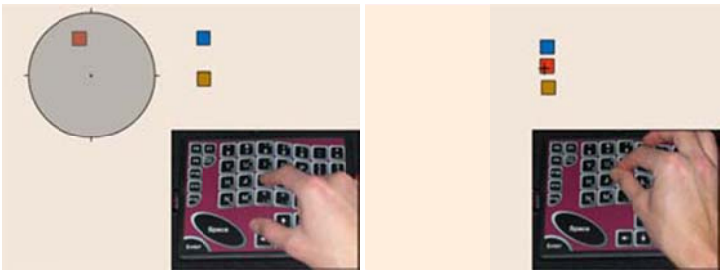


Figure 8: "An adjustable area cursor makes it easy to select isolated targets (left) while seamlessly allowing for precise selection of individual targets (right)". (Moscovich, 2006)

Third MFCT in Figure 8 "extends the idea of area cursors by allowing the user to control the size of the cursor's activation area (…) which is proportional to the span of the fingers". To select small isolated targets, users "simply spread their fingers and move to the vicinity of the object. Users can select a specific object in a crowded area by bringing their fingers together. (…) Users can easily grab ad-hoc groups of adjacent objects, these groups are simply determined by the radius of the cursor" (Moscovich, 2006).

Principles of multi-touch interaction

"Studies have shown symmetric bimanual methods to be useful for various manipulation tasks. Some of these tasks may be more easily accomplished using two fingers of one hand." For example "adjusting a rectangle to tightly enclose an ellipse, the user may need to focus on each side individually" (Moscovich, 2006).

"Initial explorations indicate that while users may be able to coordinate the motion of two fingers better as they approach the target, they perform the final matching better using two hands, since it is easier to separate the degrees of freedom. For a more integrated task, such as simultaneous rotation and translation, using one hand may have some advantages. The ability of a user to coordinate the motion of two hands depends on different factors than does finger coordination" (Moscovich, 2006).

Remarks

For some manipulation tasks or for some interactions, the symmetric bimanual methods are more appropriate, like manipulations involving two different targets at the same time. A single hand (dual finger) method might have some more advantages if used for more integrated tasks, such as simultaneous rotation and translation.

The cursors presented before are a way of giving the user feedback and an adaptation of them could be used to give useful feedback in a multi-touch-screen environment. Obviously

since the user will have his fingers on the screen, the cursors would have to be rethought in a way that they would not be obscured by the user's hands and fingers.

2.2.4.2 *Touch & Write*

Liwicki et al. (Liwicki, Rostanin, El-Neklawy, & Dengel, 2010) have developed a rear-projection tabletop device that allows users to interact with gestures and also with a digital pen, and this tabletop is called Touch & Write.

"It combines the FTIR technology for touching with the Anoto-technology for handwriting. This allows an implicit switch between the modes object manipulation and content editing". The "system incorporates real time gesture and handwriting recognition. Drawn objects and written concepts can be converted to digital information immediately". To make use of the full capability of the Touch &Write table, they have introduced a functional application, the LeCoOnt that is software for concept mapping (Liwicki, Rostanin, El-Neklawy, & Dengel, 2010).

"Touching actions are used for arranging the concepts like sheets on a normal table, and to recognize gestures like zooming. Pen-actions are used for drawing, connecting concepts, and handwriting. The handwritten strokes are automatically recognized and converted into a machine-readable string", which makes it useful for other applications (Liwicki, Rostanin, El-Neklawy, & Dengel, 2010).

Remarks

"Since intuitive handwriting is allowed as an input metaphor, the creativity is furthermore supported by this system" (Liwicki, Rostanin, El-Neklawy, & Dengel, 2010). Certainly it foments the creativity, however handwriting here is supported by the Anoto-technology and it is not suitable for use on Tablet PCs. A more adequate manner of handwritten input is required.

Touch & Write is a very similar project to this one, and they have combined hand interactions with pen interactions in an interesting natural way. They separated hands and pen interactions, since the pen is used to perform the usual pen tasks like draw or write and the hands are used to do the actions that would be done with paper on a real table, such as rearrange the sheets of paper. A similar combination of pen and fingers could be used on the prototype.

2.2.5 Conclusion

From the various domains discussed above, there are two that show more promising in terms of innovating interaction with Pen and Fingers; they are Sketching CAD and Sketching CASE tools. Other areas, despite being on the scope of this project, had some aspects that were not too adequate or were less attractive.

In the case of navigation in Digital Documents, existing effective and intuitive techniques are already well established and spread, not leaving much space for improvements. Also relating to the use of both Pen and Fingers in this domain just does not seem justifiable.

Digital Arts was a very exciting area at first. But then, after realizing the sensory experience required by the user, and the fact that the technology which provides that experience is not easily accessible, this area was also not deemed adequate for this project.

So that sorted Sketching tools as the elected area for the project. Not only because the others were rejected but as mentioned before Tablet-PCs are a suitable environment for sketching. Also there is still very much that can be done to further develop this area, so there is a lot this project can pick up from and improve on.

Now all that was left was to decide between Sketching CAD tools or Sketching CASE tools. That was a relatively easy decision to make since this is a software engineering and interaction design project, also here at the university there is history with CASE tools related work, namely Metasketch (Nobrega, Nunes, & Coelho, 2006). This last reason could provide opportunity to do a collaborative work with other projects in parallel development, which would at least make thinks more interesting. So Sketching CASE tools became the application domain for the project.

This domain was then scoped on Human Activity Modeling (HAM) (Constantine, 2009) to simplify the development of the prototype and to keep the focus on testing the pen and fingers interactions. HAM was chosen because there has not been much support by the CASE tools for this important tool of HCI design processes, in contrast with the UML class diagram, as we will see next there are already some tools that concentrate on that.

Eventually the domain got more focused on the Participation Map from HAM (Constantine, 2009) as the prototype got developed, but studies and concepts still used other diagram types form HAM and UML respectively the Performance Map and the Use Case. Participation Map was chosen mainly because there was a possibility to collaborate with a colleague that was doing his project MSc Dissertation on CASE tools as well, and his prototype fits quite well as the tool that comes after sketching. It is where the models produced here with sketching can be further refined and worked over to create either documentation or generate other types of outputs.

2.3 CASE tools

Computer-Aided Software Engineering (CASE) refers to software used for the automated development of software systems with intention of resulting in high quality, defect free and maintainable software products. CASE can also refer to the methods used together with automated tools in the software development process for the development of information systems. “CASE functions include analysis, design, and programming. CASE tools automate methods for designing, documenting, and producing structured computer code in the desired programming language” (Wikipedia Foundation, Inc., 2004a).

At the University of Michigan, the ISDOS project started exploring the concept. Daniel Teichroew with “his insights into the power of meta-meta-models” (Wikipedia Foundation, Inc., 2004a) first introduced the PSL/PSA, in 1976 (Ley, 2011), which “was the best-known output form the ISDOS project” (Darnton, 2001).

Then emerged the logical extension to the DBMS directory and so “by extending the range of meta-data held, the attributes of an application could be held within a dictionary and used at runtime” (Wikipedia Foundation, Inc., 2004a), which later gave birth to the Model Driven Execution (MDE). The problem was that meta-data was not represented graphically.

So the term CASE first appeared in Michigan in 1982, with an integration of graphics and text editor in GraphiText, developed by a software company called Nastec Corporation of Southfield. GraphiText “was the first microcomputer-based system to use hyperlinks to cross-reference text strings in documents—an early forerunner of today's web page link. GraphiText's successor product, DesignAid, was the first microprocessor-based tool to logically and semantically evaluate software and system design diagrams and build a data dictionary” (Wikipedia Foundation, Inc., 2004a).

Improvements on storing details of the data types and related processes fueled the development of data dictionaries and graphic tools. “Eventually, graphic tools integrated with data dictionary databases to produce powerful design and development tools that could hold complete design cycle documents”. In these cycles was included the “tests on the software in a continuous process of verification and validation”, which “significantly reduced the tasks that were to be undertaken during the testing phase of software”. Then in late 80s decade, “automatic generation of code took a more central stage. CASE tools that generated code were developed for various fourth generation programming languages” and where further improved in the 90s (Bett, 2007).

90s was the decade of CASE workbenches and environments. In those similar tools were brought together and integrated to solve the increasing problem of CASE tools integration and interoperability. Components then become increasingly more reusable and user interfaces friendlier to reduce the learning curve of software engineers and programmers. With the growing coverage of the software development paradigm by CASE tools they started to be used throughout the entire software development life cycle, including project management (Bett, 2007).

With the turn of the century “software engineering continues to become more complex with the availability of various software platforms for development. The desire to seamlessly shift back and forth from different platforms has forced the CASE tools to evolve further to meet the needs of the industry. CASE tools have therefore been developed to operate on a baseline standard such as the .NET framework”. Nowadays “software reengineering is also becoming a very important methodology and CASE tools have been developed to provide for this need” (Bett, 2007).

CASE software can be classified into 3 categories (Wikipedia Foundation, Inc., 2004a): Tools, Workbenches and Environments. This project’s prototype is a Tool because only supports a specific task of the software process, which is the initial sketching of user activities.

Further discriminatory classification puts it in the Upper Life-Cycle Based CASE tool bin. Is there because the activities supported (HAM) are more concerned with “concept-level products and ignore the design aspect” (Wikipedia Foundation, Inc., 2004a).

Within the CASE tools there are those which produce diagrams of various kinds. Now these diagrams can be stored in different ways for different purposes. So some tools produce models that can be ported to other tools and further refined or explored, and other tools just produce the diagram itself to serve as an artifact in the project documentation.

Modeling tools

Modeling tools are the ones that produce models that can be further worked on other tools or that can be engineered into other forms of artifacts. For instance a modeling tool could take a model of an UML class diagram and generate code for the classes described.

To be able to do that, the models produced are usually in a standard formal language, for instance MOF (Object Managment Group, Inc, 2011) (Wikipedia Foundation, Inc., 2002). This way these models can easily "be exported from one application, imported into another, transported across a network, stored in a repository and then retrieved, rendered into different formats (including XMI, OMG's XML-based standard format for model transmission and storage), transformed, and used to generate application code" (Object Managment Group, Inc, 2011).

Non modeling tools (diagram editors)

These tools differ from the previous ones because they do not produce models that can be used in the previews stated manners. They can only use their artifacts with themselves, and usually export those artifacts to an image format. Their purpose is one of documenting product specifications and/or for development teams to use as reference.

As seen before in the domain research this project had the possibility to be based on the MetaSketch (Nobrega, Nunes, & Coelho, 2006) which is a modeling tool that uses MOF to represent the models. So in that perspective the prototype could be further classified as a modeling tool.

Next we take a look at a few of the existing tools and study a bit their User Interfaces.

2.3.1 Existing tools

This section is very similar to the section of the domain research as we will be taking an overview of some tools that already exist and are within the selected domain. The first tools will be the traditional GUIs with widgets and toolbars, the latest ones will be sketch based interfaces. This way we can easily see the differences between the two interaction styles.

The interface will not be the only presented aspect of the tools as each type has its own individualities and challenges.

Microsoft Visio 2010

Visio supports the creation of UML diagrams, and it produces models that can be explored in the Model Explorer shown in Figure 9 on the top left panel. Several data types are supported by the models, such as C#, C++, IDL and VB Data Types. The several UML stencils are below the Model Explorer in the Shapes Panel. Elements can be dragged to the drawing area in the middle. This is a very typical GUI interface for this domain.

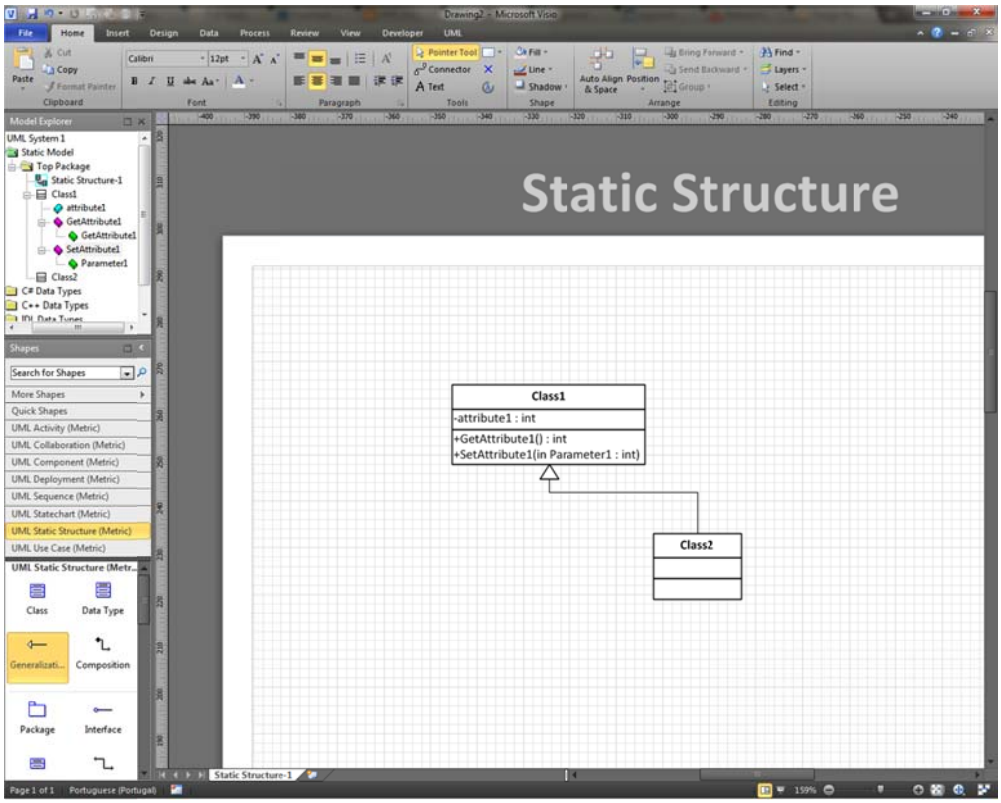


Figure 9: Microsoft Visio 2010, main UML interface.

Besides saving models in image formats, in the previous version Visio 2007, an update (Visio Team, 2009) was launched that supported exporting the models to XMI format. However in this version no exporting functionality was found.

Visual Paradigm for UML

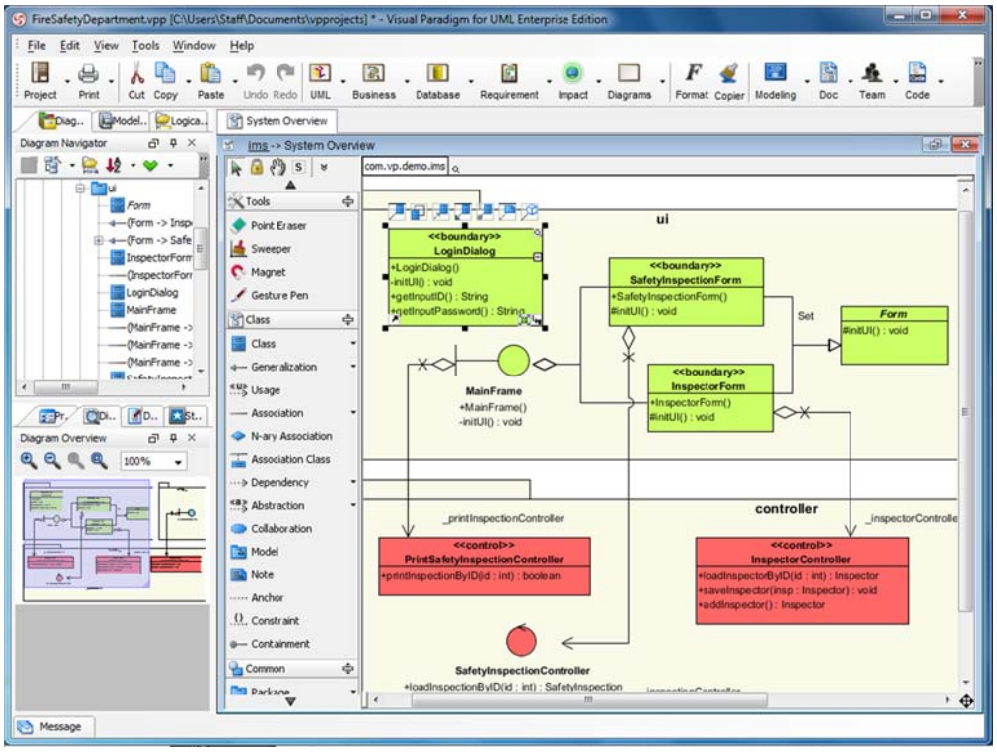


Figure 10: Visual Paradigm, UML class diagram. (Visual Paradigm International, n.d.)

Among other domains, such as Database modeling, this CASE tool fully supports the creation of UML diagrams and models. Much like Visio this tool has a model explorer tab that can be seen in the top left tab panel in Figure 10, it also has a diagram navigator. In this case the stencils are located in the same panel/frame as the diagram. And the Properties panel is located in the bottom left tab panel, together with the Diagram Overview and other panels.

The models can be saved to image formats and also can be exported/imported to and from XML; code can be generated to Java (Visual Paradigm International, n.d.).

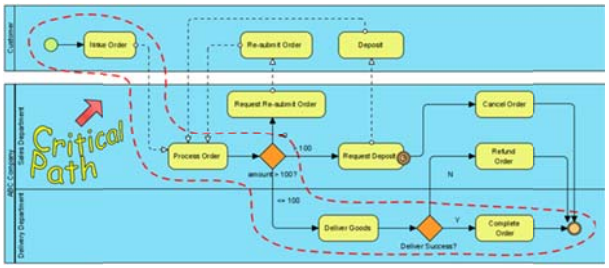


Figure 11: Freehand Shapes in Visual Paradigm. (Visual Paradigm International, 2009)

There is an interesting shape support that allows the user to create general shapes for annotations, either by freehand sketching or solid shapes. An example of this feature is in Figure 11, the red dashed line, the red arrow and the yellow text "Critical Path" are all annotations. A similar feature was developed in the prototype as we will see further in chapter 5 The Prototype.

Meta Sketch

As mentioned before the Meta Sketch (Nobrega, Nunes, & Coelho, 2006) uses MOF (Object Managment Group, Inc, 2011) to represent models, and to explore them, it also uses a Model Explorer that is located on the left panel, shown in Figure 12. The diagrams are represented in the middle sketching area, and the tools and properties are on the right side by default but can be moved by the user, as in other editors.

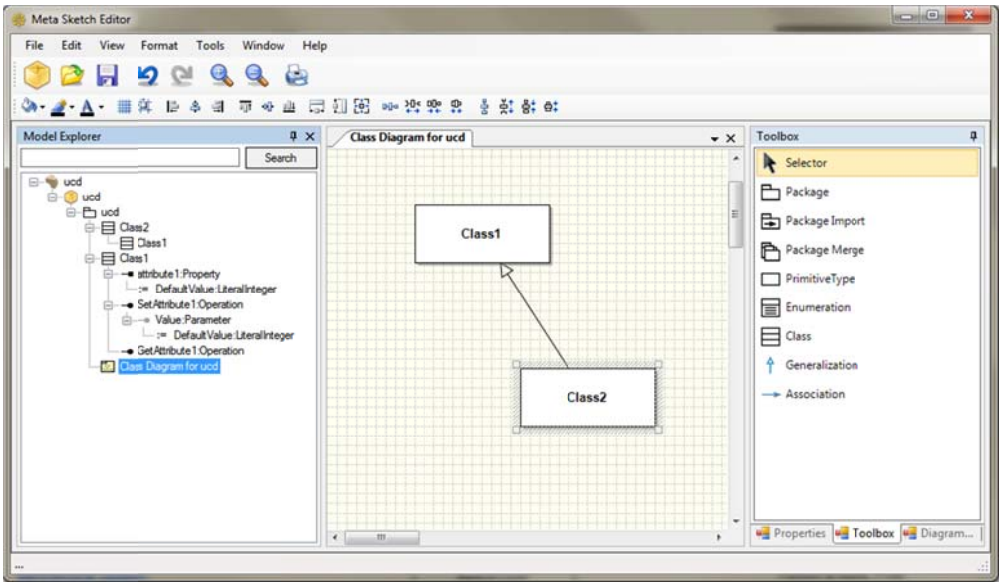


Figure 12: Meta Sketch Editor, Class Diagram for User Centered Design.

Meta Sketch not only supports UML class diagrams but also integrates "different HCI notations into the UML 2.0", such as Concurrent Task Trees (CTTs) and canonical Abstract Prototypes (CAPs). This provides "semantically sound metamodeling extensions that could

leverage the MDD paradigm" and "support automatic generation of interactive applications" (Nobrega, Nunes, & Coelho, 2006).

Later OCL was integrated by Matos (Matos, 2008) into Meta Sketch, giving it the ability to describe the semantic aspects of the modeling languages thus allowing it to semantically verify its models (Matos, 2008).

Finally the models can be exported as XMI and CMOF format to be used on other tools; they can also be imported from those formats.

AgroUML

AgroUML has the interface a bit different than the previous tools, because it has an extra feature, a "to do" list located in the bottom left panel shown in Figure 13. We can also see in the bottom right panel the "details of the selected object in the diagram or the selected «to do» item", other tools presented the proprieties on the side panels. Otherwise the interface is similar to the previous, as the model of the diagram is presented in the top left panel and the diagram is presented in the right panel (CollabNet, Inc., n.d.).

The models use OCL to define model constraints and can be exported to XMI; code can also be generated from the models in several languages, namely Java, C++, C#, PHP4 and PHP5. AgroUML can also reverse engineer form Java to create visual representations and models of the code (Parenteau, 2009).

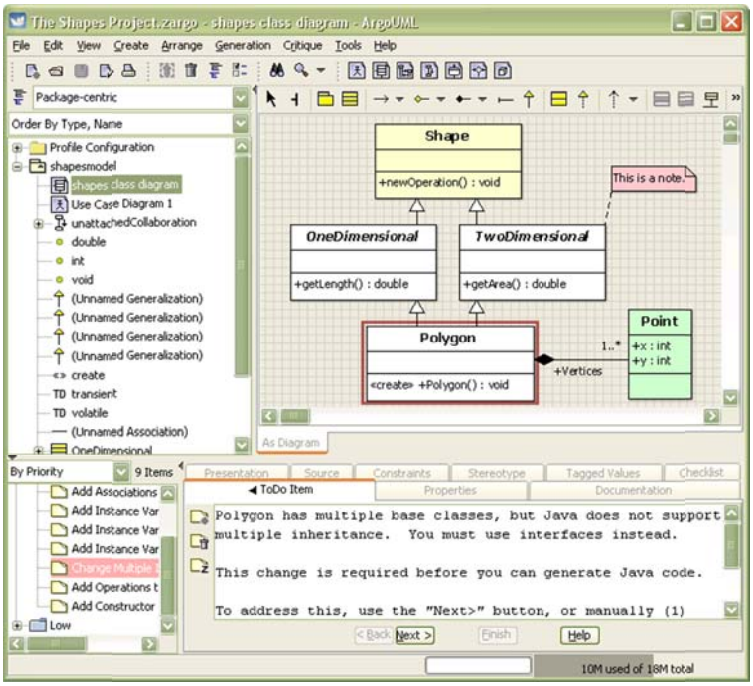


Figure 13: AgroUML General interface. (CollabNet, Inc., n.d.)

Tahuti

Tahuti is a different CASE tool than the ones shown above, because it is a "multi-stroke sketch recognition environment for class diagrams in UML". It has two visual modes as "users can draw and edit while viewing either their original strokes or the interpreted version of their strokes engendering user-autonomy in sketching", but users preferred the interpreted view (Hammond & Davis, Tahuti: A Geometrical Sketch Recognition System for UML Class Diagrams, 2002). This dual-view (shown in Figure 14) has the freedom of sketching in paper and the processing power of diagram editors.

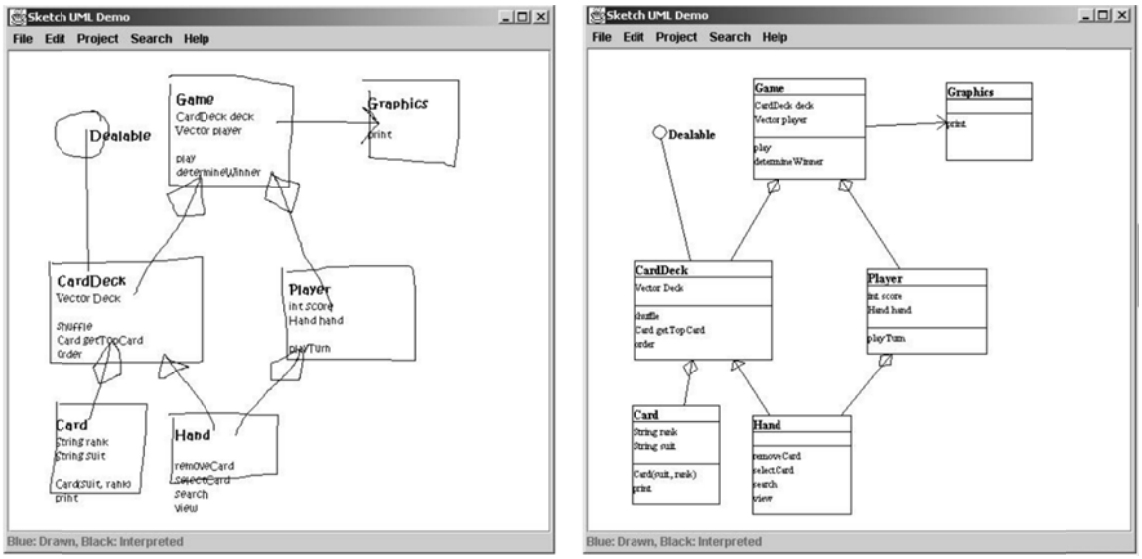


Figure 14: Tahuti - the original user strokes view (left); the interpreted view (right). (Hammond, Gajos, Davis, & Shrobe, 2002)

The interface for this system as seen in the Figure 14 is very open and non-intrusive to allow the user as much freedom as he needs to foster his creativity, which we have seen in 2.2.1.1 The Sketching Paradigm is essential when in the initial stages of the design.

"The system is based on a multi-layer recognition framework and recognizes objects by their geometrical properties, rather than requiring that the user draw the objects in a pre-defined manner". That approach maintains the recognition accuracy and allows users to sketch naturally (Hammond & Davis, Tahuti: A Geometrical Sketch Recognition System for UML Class Diagrams, 2002).

SUMLOW

This tool is similar to the previous one as it is based on sketch input. SUMLOW is “a Unified Modelling Language (UML) diagramming tool that uses an E-whiteboard, pen-based sketching interface to support collaborative design”; it “allows designers to sketch UML (…) mixing different UML diagram components, free-hand annotations and hand-written text” (Chen, Grundy, & Hosking, 2003).

As with Tahuti, the hand-drawn diagrams are preserved and they support “manipulation of the diagrams using pen-based actions”, which again lets users sketch naturally. Then the “sketches can be «formalized» to computer-recognized and drawn diagrams, and exported to a 3rd party CASE tool” (Chen, Grundy, & Hosking, 2003).

As can be seen in Figure 15, the interface is similar to the previous tool, it is very unobtrusive and focuses on the sketching, hence the sketching area portion of the interface. The elements and the text are “constructed and manipulated using pen-based input. The freedom from heavily-enforced modeling constraints and flexible annotation facility in this environment encourages exploratory and collaborative UML-based software design” (Chen, Grundy, & Hosking, 2003).

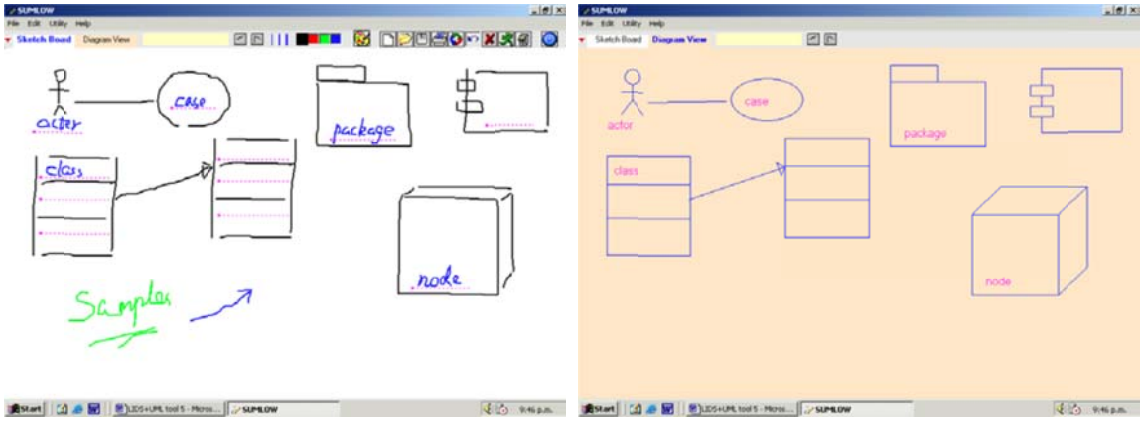


Figure 15: SUMLOW's various recognised UML constructs - sketch view (left) and diagram view (right). (Chen, Grundy, & Hosking, 2003)

Recognition has a few problems with some sketch elements, "particularly with the single-stroke text recognition algorithm" but still has an average of 92% recognition rate for sketch elements. Another issue is the "learning curve for some gestures" can be somewhat steep (Chen, Grundy, & Hosking, 2003).

However SUMLOW has a limitation of sketches being only forward engineered to UML, and not able to be reversed engineered or imported back to sketch format for further manipulation (Chen, Grundy, & Hosking, 2003).

2.3.2 User Interface

From the tools studied here and from section 2.2 Related Work and Domain Research, is possible to extrapolate the common aspects of the user interfaces for both the traditional interface CASE tool and the sketch based interface CASE tool. Figure 16 shows wireframes that illustrate those common aspects.

It is immediately apparent the differences between the two interfaces. The traditional is cluttered with information and tools and functionalities; where the sketched based is a very "clean" and simple interface which focuses on sketching area and has the toolbar as a support and not as its main interaction point.

Figure 16: Common interface wireframe for traditional CASE tools (left) and for sketch based (right).

In the traditional interface all the panels and functionalities are there at a click's distance (the properties panel is not in the wireframe because its position usually varies from tool to tool). That is good when there is a need to access information about an element of the diagram or preform some kind of manipulation, but if the goal is to create new data or new

elements, all that availability of functionality and information is a distraction to the user and can cripple his creativity.

Hence the sketch interface has a very wide sketching area with nothing but blank clean «paper» (space) for the user to express his creativity and release his imagination to solve the problem in hands. The toolbar has only the essential tools to support the sketching activities and is usually small in size as to maximize the sketching area.

2.4 Technology

Touch technology as we know it today is present in many areas and spread through many devices, “widely used on ATM machines, retail point-of-sale terminals, car navigation systems, medical monitors and industrial control panels” (Belis, n.d.), and on handhelds.

Nowadays touch technology is usually associated with touch-screens, but it is present in other forms and device types such as touch-pads which are commonly used on laptops. There is a very important distinction here, “it is a difference of *directness”* as Buxton explains (Buxton, "Multi-Touch Systems that I Have Known and Loved" at billbuxton.com, 2007). Touch-screens or touch-displays are actual screens that allow the user to directly interact with (touch) what he sees and a touch-pad or a touch-tablet is a surface that captures touch input but does no present the interface itself or it is not overlaid on a screen that does so.

> So for this document the term Touch-screen represents *“an electronic visual display that can detect the presence and location of a touch within the display area”.* Touch here means either a finger or a passive object, such as a stylus *(Wikipedia Foundation, Inc., 2004)*. *Adding further to this definition, depending on the characteristics of the device (touch-screen) itself, a touch-screen could detect other aspects of the touch, such as pressure, size, orientation, etc.*

However touch devices were not always as spread and accessible as today; as many other technologies, touch started with industry applications and only later came to the more personal and mobile use.

Actually “touch-sensitive control devices pre-dates the age of the PC” (Buxton, "Multi-Touch Systems that I Have Known and Loved" at billbuxton.com, 2007) and the touch-screen. Hugh LeCaine et all, started experimenting with touch-sensitive capacitive-sensors on synthesizers to control sound and music. The touch-screen only first showed up in 1965 by E.A. Johnson at the Royal Radar Establishment, Malvern, UK; and its first application was an air traffic control station, as suggested by Johnson in 1968 (Belis, n.d.).

In the 70s several techniques to capture touch emerge, but all were single-touch and not pressure-sensitive. In 1972 at the University of Illinois created the terminal PLATO IV, a computer assisted education system, which had been in development since the 60s; PLATO used infrared to capture touch and invented the flat panel plasma display (Buxton, "Multi-Touch Systems that I Have Known and Loved" at billbuxton.com, 2007). Sam Hurst develops the "Elograph" touch sensor back in 1971, it was not a transparent like the modern sensors, but was a very important milestone. Later in 1974 Elographics (founded by Sam Hurst) launches a touch-screen with fully transparent sensor. Then in 1977 he developed the resistive

touch screen technology, which is widely used today. In the same year, financed by Siemens, Elographics produced "the first curved glass touch sensor interface, which became the first device to have the name 'touch screen' attached to it" (Belis, n.d.). In 78 Herot and Weinzapfel demonstrated a screen that “could sense 8 different signals from a single touch point: position in X & Y, force in X, Y, & Z (i.e., sheer in X & Y & Pressure in Z), and torque in X, Y & Z” (Buxton, "Multi-Touch Systems that I Have Known and Loved" at billbuxton.com, 2007) as illustrated in Figure 17. This was an important display of the potential richness of touch screen interaction.

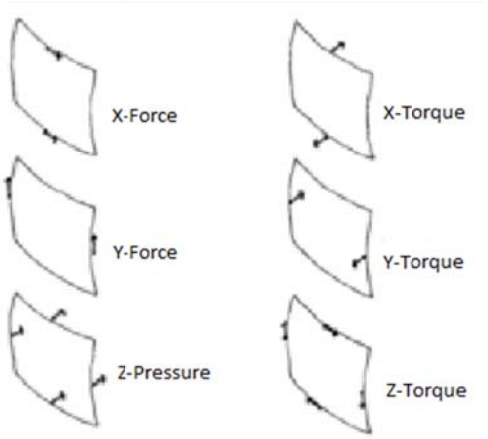


Figure 17: Herot’s and Weinzapfel’s One-Point Touch Input of Vector Information, force and torque illustration. (Buxton, "Multi-Touch Systems that I Have Known and Loved" at billbuxton.com, 2007)

The 80s was a decade marked by multi-touch. In 1982 Nimish Mehta at the University of Toronto presented the Flexible Machine Interface, a camera based multi-touch system that “consisted of a frosted-glass panel whose local optical properties were such that when viewed behind with a camera a black spot whose size depended on finger pressure appeared on an otherwise white background” (Buxton, "Multi-Touch Systems that I Have Known and Loved" at billbuxton.com, 2007); it was possibly the first multi-touch system for human input. In 1983 Myron Krueger invents the Video Place, a video based system that tracked the hands and enabled multiple fingers, hands, and people to interact using a rich set of gestures, and it could also be configured in table and wall. Then in 1984 Bob Boie at Bell Labs creates the first multi-touch screen with a “transparent capacitive array of touch sensors overlaid on a CRT” (Buxton, "Multi-Touch Systems that I Have Known and Loved" at billbuxton.com, 2007). Several other efforts were made on multi-touch research, such as capacitive touch tablets (pads), optical frames to overlay screens and multiple “hard devices” for simultaneous input such as trackball, mouse, joystick, etc. It was in this decade that the first tablet pc was developed, but we will talk a bit more about it further down the chapter.

In the 1992 the first smart phone, Simon was developed by IBM and Bell South, it used a single-point touch-screen user interface; in the same year Wacom launched their multi-device/multi-Point tablet digitizers, that could simultaneously detect a stylus with several degrees of freedom and other devices (Buxton, "Multi-Touch Systems that I Have Known and Loved" at billbuxton.com, 2007). “In 1993, Apple released the Newton PDA” and “in 1996, Palm entered the PDA market and advanced touch screen technology with its Pilot series” (Belis, n.d.). Tangible interfaces also started to appear in mid 90s, and other tablets with multi-touch sensing capabilities were later introduced.

The 2000s is when the real explosion of touch interface happens, and it could be defined by the touch-screen smart phone as Apple launched its iPhone in 2007 (Belis, n.d.). Regardless other touch environments are becoming more and more widespread such as interactive walls and table-tops (i.e. Microsoft Surface in 2007) (Buxton, "Multi-Touch Systems that I Have Known and Loved" at billbuxton.com, 2007), and tablet-PCs (including slates and pads). And

with the advent of multi-touch technology new multi-touch interactions and techniques have been and are being explored, so one can only expect in the near future more touch environments and more application of the NUI interaction style.

Several different technologies that support touch were mentioned above, being the most commonly used today the capacitive in mobile devices and optical IR imaging with the larger devices. But with the first mobile touch devices (tablet-PC, PDAs) the most common technology was the resistive panel which was better operated with a stylus. However there are more types of touch technology that uses things like ultrasonic waves or mechanical energy to detect touch (Wikipedia Foundation, Inc., 2004).

2.4.1 The Tablet-PC

A tablet personal computer is a mobile computer that uses a touch-screen as display. So it supports direct input on its display, either stylus or fingers, or both. There are several types of tablets, being that generally they could be categorized as slates or convertibles (Everet, 2010).

Types of Tablet Computers

A slate is basically a flat display with an embedded computer in it that can be used like a paper notebook. It relies on the touch-screen for text input as it does not have a keyboard, accessories like a keyboard or a mouse can be plugged in so it can be used as conventional computer. Its purpose is extreme portability while having many if not all the functionality of a desktop (Everet, 2010) (Malysa, 2010).

Figure 18: The HP TouchSmart tm2, a convertible tablet PC. (Hewlett-Packard Development Company, L.P., n.d.)

The convertible type looks like a normal laptop, because it is one, but it has an extra characteristic that distinguishes it from all other laptops; has a touch-screen that can fold or slide so the laptop becomes a tablet. It has all the capabilities of the slates but the extra laptop keyboard and mouse, and also the all-important power. This obviously makes the convertibles a bit heavier and thicker than the slates (Everet, 2010) (Malysa, 2010).

The tablet can be further classified as other designs introduce some differences in the concept and make a type of their own. For instance the booklets, which "are dual-touchscreen tablet computers that fold like a book" (Wikipedia Foundation, Inc., 2006); they are like two slates put together. Or the hybrid, which "share the features of the slate and convertible by using a detachable keyboard that operates in a similar fashion to a convertible when attached"; i.e. the keyboard can detach like a normal keyboard accessory for a slate,

Figure 19: Microsoft's Courier Booklet. (PAPERBOY, 2009)

but when its attached to the touchscreen it can rotate and “allow the tablet to rest on it like a convertible” (Wikipedia Foundation, Inc., 2006).

There is still the rugged type, which can be any of the previous types of tablet, but is usually more durable by “resisting heat, humidity, and drop/vibration damage”. Its application is more oriented to field work, and for that they are built either with little to none moving parts or with shock resistant ones, also are usually encased in resistant metal and rubber shells. (Wikipedia Foundation, Inc., 2006) (Malysa, 2010).

History of the Tablet

“The concept of capturing handwriting electronically goes back to a patent from 1888” (Holwerda, "Tablets: Sought by Nobody, Hyped by Everybody" at OSnews, 2009) which was “related to the telegraph”. Nonetheless in 1956 the Stylator tablet was “the first occurrence of a pen-style input device replacing a keyboard”. But RAND tablet in 1964 was “a more well-known tablet input device” (Holwerda, "Tablets: Sought by Nobody, Hyped by Everybody" at OSnews, 2009). However both of them where not the tablet configuration as we know today, they supported electronic handwriting input though a specialized pen and a tablet slate.

The Tablet-pc was initially “conceptualized by Alan Kay in the late 1960s and early 1970s” and already introduced “several ideas that would become commonplace only decades later”. For instance it would “connect wirelessly to centralized information storages,” and at the time it was designed with a hardware keyboard as in Figure 20. But roughly 40 years ago, Kay “describes a multi-touch display”, which could place any keyboard arrangement anywhere on the screen and display the current text font directly on the keys. This screen would capture touch using “four strain gauges mounted under the corners of the panel”. However “apart from printing the word "stylus" in one the illustrations, there is no further mention of it in the article” (Holwerda, "A Short History of the Tablet Computer" at OSnews, 2010), Kay’s DynaBook dissertation (Kay, 1972).

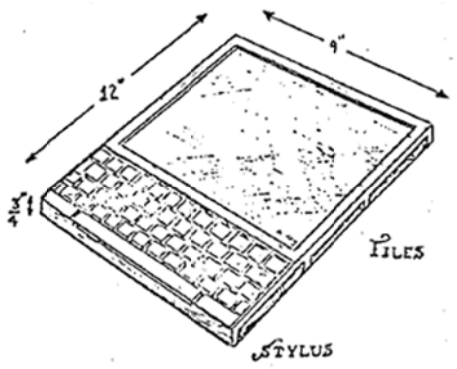


Figure 20: Dynabook conceptual sketch. (Holwerda, "A Short History of the Tablet Computer" at OSnews, 2010)

“The first, actual tablet computer: a portable device which uses a touch screen for all its input” and also used a stylus, was the GRiDpad from 1989. In 1993 Apple stated shipping the Newton MessagePad, they “originally intended the Newton to be a tablet-style computer, but during its development, it turned into a PDA” (Holwerda, "Tablets: Sought by Nobody, Hyped by Everybody" at OSnews, 2009). Then in 2002 “Microsoft and its PC maker” partner HP, “deliver their own Tablet PC” (Holwerda, "A Short History of the Tablet Computer" at OSnews, 2010) and “introduced the Windows XP Tablet edition and started its entry into touch technology” (Belis, n.d.). Tablets saw an important improvement when in 2007 Microsoft presented its ThinSight that was capable of sensing multiple points and objects and was thin enough to incorporate it into a tablet-PC. Also in 2008 N-trig came up with its dual digitizer that could also detect both fingers and a stylus, and since then other manufactures have

developed their one dual digitizers (Buxton, "Multi-Touch Systems that I Have Known and Loved" at billbuxton.com, 2007).

Then in 2010 Apple announced and released its iPad tablet, to which many others followed. It did not include a stylus nor it supported one, which was a runaway form the traditional tablet, but it fully supported multi-touch interaction (Wikipedia Foundation, Inc., 2009). This set the trend for the most recent tablets as Google also launched its own operating system, the Android which was first used in smartphones. Most of the recent tablets use an installment of Google's OS as its open source, in contrast with Apple's mobile iOS that is proprietary and it is used on other Apple's mobile devices.

2.5 Summary

After having the initial research, on touch environment, several aspects and nuances were found that are well established and that give tremendous support in terms of guidance to this type of project in the sketch application domain. This is in reference to pieces of knowledge, the guidelines and the requirements, and the inside knowledge that was dispersed throughout all the tools studied and the literature. All of those contributed to a clearer insight of the domain and brought great influence in the design decisions taken throughout the project.

As mentioned initially the project did not have a defined domain of application, so the research for that area was done to find one, and quite well a perfect domain for the project was found. In that research several tools where studied with the purpose of evaluating their fitness for the project and form those tools some requirements were found that could be applied in this case as well, they are presented in section 3.4 Requirements.

We looked into CASE tools to find what had been done so far and to study a bit the user interfaces and the interaction styles used. We have seen that annotation using a pen had already been used in some of the tools that still used Windows, Icons, Menus and Pointers (WIMP) interaction style, and the pen had become the primary input device in some of the other CASE tools studied, which were trying to move away from the traditional WIMP and into a more natural interface type.

However from the tools studied we only saw WIMP and strictly pen based sketch interfaces and not interfaces that made used of the users natural hand and fingers skills and abilities, such as pointing or moving objects with the finger. This created an area where NUI could be explored and this project makes use of that opening.

From the literature we have also seen that paper sketching is still the initial resource of developers when starting a new design and that it is possible to improve greatly from it. Since HAM can be considered somewhat an early diagramming domain still, certainly when compared with UML, there is an opportunity for this project to try to innovate both in the aspect of interaction as well as in the aspect of CASE tools.

3 Problem Analysis

As mentioned in section 1.2 Problem and Objectives, the problem was how to combine effective and efficient pen and finger interactions with a touch-screen in a sketch application domain, CASE tools in this case.

For that a better understanding of the application domain of the system is needed. So there was conducted a study of the current way of doing things in software development process; that is the use of paper sketches to create the initial concepts when designing a new system or application, and then the production of the "definitive" models and diagrams of those concepts in a CASE tool on a computer.

This study begins with the first step of creating the initial draft models of the activities users perform when conceptualizing a new application. The expected result was a better understanding of those activities which could serve as foundation to a reasonably detailed description of the tasks users would make in the prototype, and a set of requirements needed to guide the design and development of the prototype.

This description requires some more information about the actual prototype which meant simultaneously starting the initial conceptualizations of the system and the interaction affordances that would be supported by the prototype interface. Those will be presented next in chapter 4 Concept. The knowledge gained form this study will in its turn influence heavily on the concept of the system. This emphasizes the cyclic nature of the research process used, in which what started as an input to another step becomes part of the outcome of the very same process.

With the draft models done there was conception of a plan to conduct the observations and interviews, which is fully described in Appendix 1 along with all the material from the study. A scenario was composed in which the observations and interviews would be based upon, and so a list of tasks was then devised for the user to perform on the observations and a questionnaire was created to follow up on the observations. Those, including the scenario, where then organized into a task sheet that was given to the users at the beginning of the observations; they can also be found in Appendix 1.

After the initial observations were conducted the scenario was revised and shortened as it was too long and was making the whole process a bit lengthy. That gave origin to another shorter version of the task sheet and more observations were conducted which made this somewhat an iterative process.

3.1 Models

The modeling was based on the Human Activity Modeling processes described by Constantine (Constantine, 2009), however not all models or aspects of a model where used since this was an exploratory modeling or because they were not considered necessary for this project.

First model is about the contextual aspects of modeling of a system in the early stages of the software process, this means a focus on the activities that are done by developers and

designers on those steps. The second model is focused on who participate on those activities, and finally the third model is all about the tasks and actions and their relationship with the activities within which they are performed.

3.1.1 Context

The context is about the activities that are done by developers and designers when they are sketching and creating diagrams, the environment that they are performed in, their relationships and their descriptions.

The first artifact (in Table 2), is a compilation of the involved activities.

Table 2: Activity Inventory.

Activities	AC1 - Modeling an application AC2 - Sketching on paper AC3 - Editing a diagram AC4 - Annotating on a diagram AC5 - Create a new diagram AC6 - Annotating on a paper sketch

Next is an activity map in Figure 21 that describes the relationships between the activities.

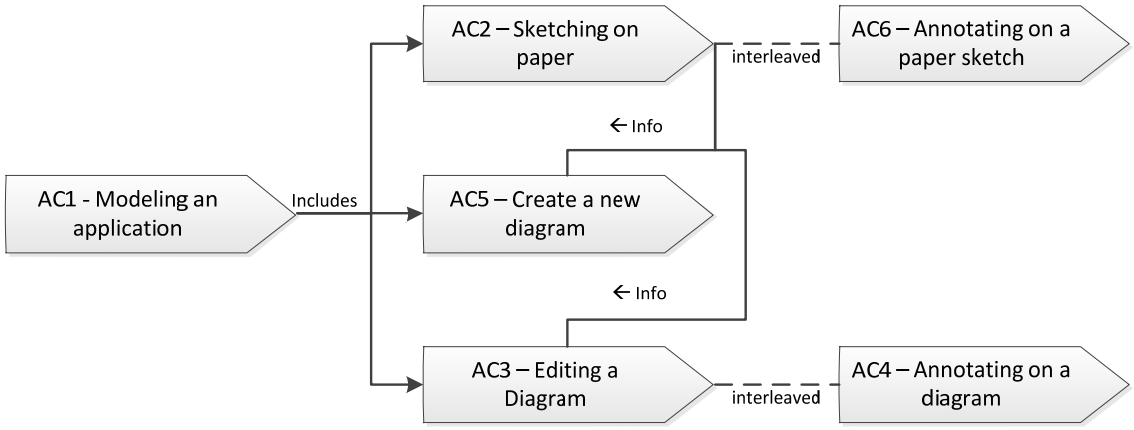


Figure 21: Activity Map.

Here we can see that AC1 is sort of the primary activity as it contains most of the remaining activities. When sketching on paper people often take notes on the sketch itself as they are creating it, so AC2 is concurrent or interleaved with AC6. The same happens with AC3 and AC4, but this obviously depends on the functionalities of the diagramming tool being used, some do not support any kind of annotation.

After the initial sketches are created, comes the painstaking task of passing them and the concepts within them to a more formal format, so information form the sketches is passed to the diagrams as represented by the connections from AC2 to AC5 and AC3.

The next table contains the profiles cards for each activity, which are succinct and template descriptions of the activities (Constantine, 2009).

Table 3: Activity Profile Cards

AC1 – Modeling an application	
Purpose	To produce models that correctly describe the functionality, the aspect and the environment of an application
Place and Time	In brainstorming sessions, project meetings, model production tasks; Collaborative or single work over paper or digital environment; Durations may vary form a couple of minutes to consecutive hours.
Participation	All participants (A1 – Software Engineer; A2 – HCI Designer; R1 – Sketch Producer; R2 – Model Producer; P1 – User's Colleagues; AR1 – Sketch; AR2 – Diagram)
Performance	Contains activities AC2 – Sketching on paper, AC3 – Editing a diagram and AC5 – Create a new diagram. It is usually performed in the beginning of the design of a user interface.
AC2 – Sketching on paper	
Purpose	Creating and capturing the concepts of human activity within a particular system
Place and Time	In brainstorming sessions, project meetings; Collaborative or single work over paper; Durations may vary form a couple of minutes to consecutive hours.
Participation	A1 – Software Engineer; A2 – HCI Designer; R1 – Sketch Producer; P1 – User's Colleagues; AR1 – Sketch;
Performance	Usually performed concurrently with AC6 – Annotating on a paper sketch; Concepts flow form this activity to activities AC3 – Editing a diagram and AC5 – Creating a new diagram
AC3 – Editing a diagram	
Purpose	Pass a sketch to a formal useful format; Update corrections and changes to a previous created diagram.
Place and Time	In project meetings, model production tasks; Usually single but can be collaborative work on a digital environment; Durations may vary form a couple of minutes to consecutive hours.
Participation	A1 – Software Engineer; A2 – HCI Designer; R2 – Model Producer; P1 – User's Colleagues; AR1 – Sketch; AR2 – Diagram
Performance	Usually performed concurrently with AC4 – Annotating on a diagram; Information flows form activity AC2 – Sketching on paper
AC4 – Annotating on a diagram	
Purpose	Quickly put down ideas and information about the diagram
Place and Time	In project meetings, model production tasks; Usually single but can be collaborative work on a digital environment; Small period of time.

Participation	A1 – Software Engineer; A2 – HCI Designer; R2 – Model Producer; P1 – User's Colleagues; AR2 – Diagram
Performance	Usually performed concurrently with AC3 – Editing a diagram but can occur independently

AC5 – Create a new diagram	
Purpose	Create the facilities need to start passing the sketch to a diagram
Place and Time	Usual during model production tasks; And usually single work on a digital environment; Takes just a couple of minutes.
Participation	A1 – Software Engineer; A2 – HCI Designer; R2 – Model Producer; P1 – User's Colleagues; AR1 – Sketch; AR2 – Diagram
Performance	Usually this precedes AC3 – Editing a diagram, but depends on the diagramming tool; Information flows form activity AC2 – Sketching on paper;

AC6 – Annotating on a paper sketch	
Purpose	Quickly put down ideas and information about the sketch
Place and Time	In brainstorming sessions, project meetings; Usually single but can be collaborative work over paper; Small period of time.
Participation	A1 – Software Engineer; A2 – HCI Designer; R1 – Sketch Producer; P1 – User's Colleagues; AR1 – Sketch;
Performance	Usually performed concurrently with AC2 – Sketching on paper but can occur independently

3.1.2 Participation

This part is focused on who and what is involved in the activities.

Table 4 is a compilation of all participants of the activities, in this case: actors, roles, artifacts and players.

Table 4: Participation Inventory

Actors	A1 - Software Engineer A2 - HCI Designer
Roles	R1 - Sketch Producer R2 - Model Producer
Artifacts	AR1 - Paper Sketch AR2 - Diagram
Players	P1 - User's Colleagues

Next is the participation map in Figure 22 that describes the relationships among the participants and artifacts.

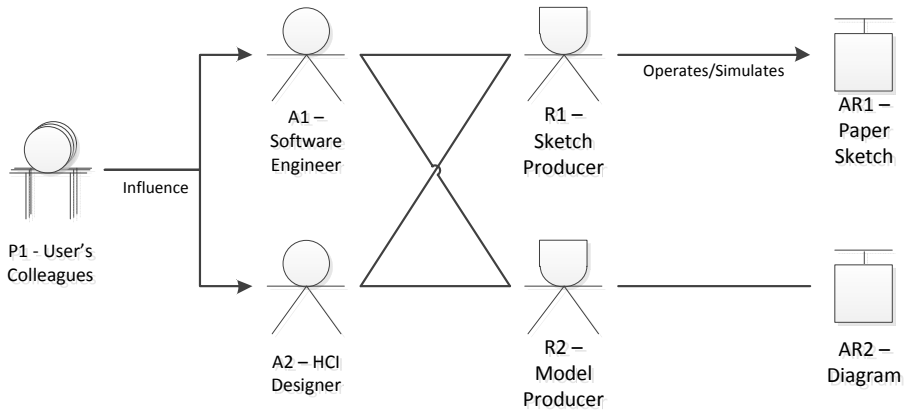


Figure 22: Participation Map.

Starting by the actors, here we have two which are A1 – Software Engineer and A2 – HCI Designer. They can be influenced by their peers who can be present in the same room as they are and start proposing ideas and concepts to the model; colleagues do not interact with the system directly so they are considered players P1 – User's Colleagues.

Actors can take roles in the system, and both actors can take both existing roles R1 and R2 because there are many variations. It can be different from project to project if there is actually an HCI Designer integrated or not, and even if there he might be responsible for the creation of the sketches and diagrams in some projects and not in others, and so on… So having this flexibility of both actors being able to perform both roles covers those variants.

Then the actor in role R1 – Sketch Producer has to simulate artifact AR1 – Paper Sketch, which means that R1 has to be the one who imposes the correct rules and constraints of the model being created as paper does not have them. With AR2 – Diagram is a different case as R2 – Model Producer does not have to simulate those rules and constraints, because the diagraming tool should already enforce them.

This is an obvious relationship but still is important to point out that R1 and AR1 are within the context of the activities responsible for all sketching tasks; namely AC2 and AC6. And R2 and AR2 fall in the context of activities AC3, AC4 and AC5.

3.1.3 Performance

The last part of the model is all about the tasks and actions and their relationship with the activities within which they are performed.

In Table 5Table 4 is a compilation of all tasks.

Table 5: Task Inventory

Tasks	T1 - Adding an element to a diagram T2 - Connecting elements from diagrams T3 - Resizing an element of a diagram T4 - Moving an element of a diagram T5 - Editing proprieties of an element of a diagram T6 - Name an element of a diagram T7 - Remove an element from a diagram

T8 - Make a copy of an element of a diagram
T9 - Name a diagram
T10 - Assign a category to a diagram
T11 - Rename an element of a diagram
T12 - Add a boundary to a diagram
T13 - Remove a boundary from a diagram
T14 - Highlight an element of a diagram
T15 - Adding a note to a diagram
T16 - Removing a note from a diagram
T17 - Draw an element on a paper Sketch
T18 - Scratch an element of a paper Sketch
T19 - Remove an element from a paper sketch
T20 - Write an element's data/proprieties on a paper Sketch
T21 - Renaming an element of a paper Sketch
T22 - Write the name of an element on a paper Sketch
T23 - Name a paper Sketch
T24 - Connect elements of a paper Sketch
T25 - Draw a boundary on a paper Sketch
T26 - Erase a boundary from a paper Sketch
T27 - Highlight an element of a paper Sketch
T28 - Writing a note on a paper Sketch
T29 - Scratch a note form a paper Sketch
T30 - Remove a note from a paper sketch

From these tasks a list of tasks was retrieved that represent the tasks that will be supported by the prototype. Further down in 4.4 Task Descriptions, that list is presented along with a Task Description card for each one.

Next is a performance map in Figure 23. In this diagram are represented the tasks that compose the activities described in section 3.1.1 Context, and their relationships. Related tasks are grouped into clusters and each cluster can relate to one or more activities. Clusters composed of tasks concerned with sketching are within sketching activities and the same for diagraming activities and clusters.

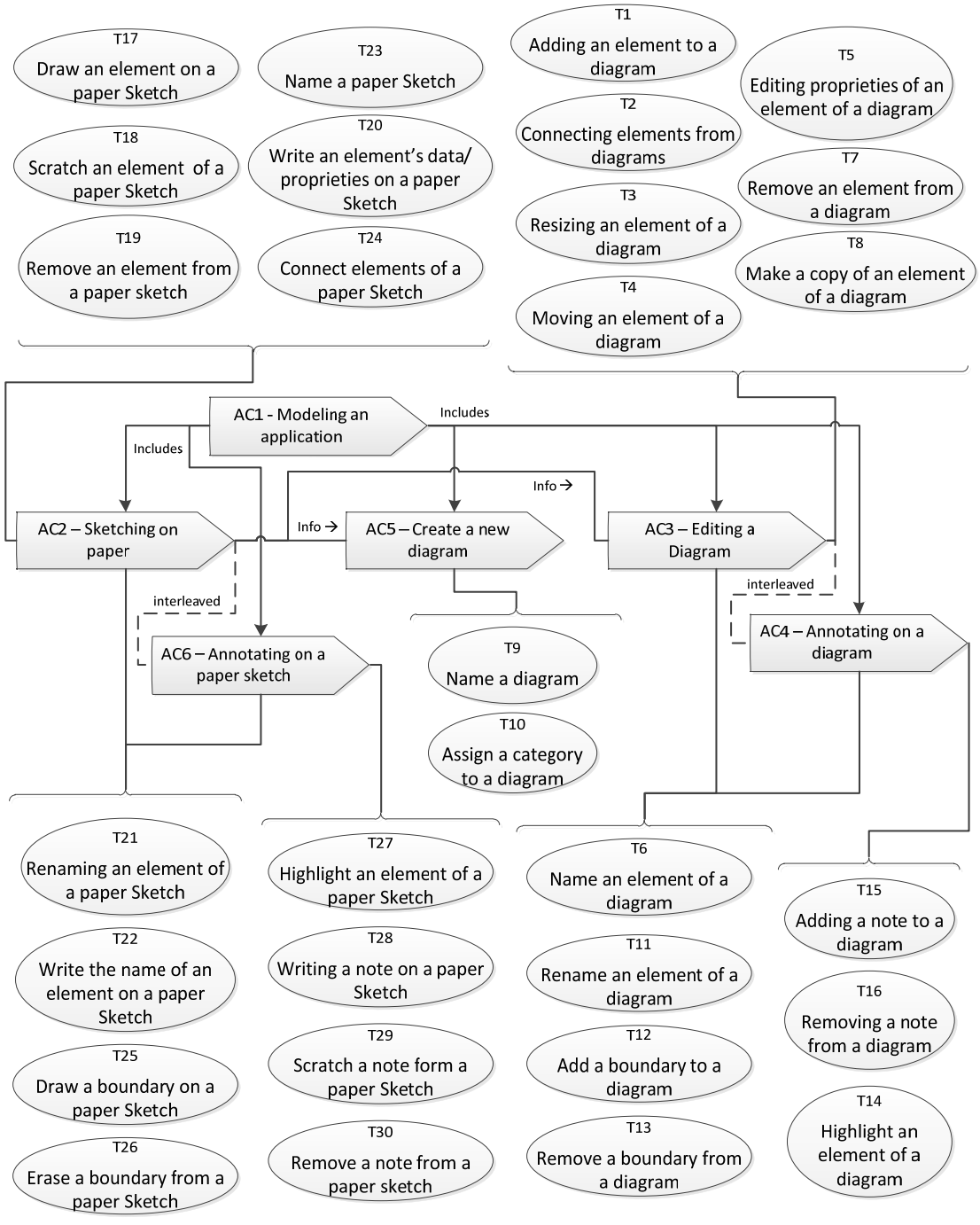


Figure 23: Performance Map.

3.2 Scenario

The scenario used was about a normal operation in an ATM and it was devised because it is a widely used example in courses about data manipulation and software engineering, and as the test subjects where software engineering students they would already be familiar with the scenario and could focus on doing the activity models. The scenario is as follows:

> *« John needs money to go to lunch. So he goes to an ATM to withdraw some cash, inserts his card and enters the PIN when asked. He has more than enough for his lunch so he proceeds by choosing "Withdraw*

money" and then enters the quantity "10€". Finally he selects "Remove card". The card comes out and he gets it from the machine. Afterwards he gets the money and then the receipt. »

This is the first scenario, scenario 2 can be found in Appendix 1. This scenario tries to have the most common elements of the activity model in sufficient quantity for the tests, being those elements the activity and the task widgets and the relationships between them.

From here as explained before the task sheets were composed and included the scenario, the task for the user and the questionnaire; they and all the rest of the observations' data can be found in Appendix 1.

3.3 User studies' results

From the observations and interviews we can draw some conclusions of some important aspects of the interactions of users with the sketch and the diagraming tools.

Users that first drew the contours of the elements they wanted to add and then wrote the names inside, usually the name overflowed the contour. So in a digital sketching environment when a contour is sketched and then the name is added, if the name is too big to fit inside the shape, then the shape should grow.

When users are drawing a new diagram they tend to under estimate the size it will have, that is because it is in the initial creation stage, so users do not know all the elements that will go in to the diagram and their relationships. What usually happens is that the last few elements added are squeezed into the available space, or are put in some available space in the sheet that breaks the spatial context of that element and the elements that it is related with. As such the "sheet of paper" (diagram area) should be "infinite" in the sense that when adding a new element, if there is no more space near the area where it should go and that area is near the visual boundaries of the diagram, one should just be able to move the diagram a bit more to get some clear space. The "sheet of paper" should not be limited by standard paper sizes as with the normal diagraming tools. That type of concern is for the step next to creating the sketch, which is producing the diagram for documentation or printing purposes.

When queried about supporting the removal of elements by gestures or the traditional menu and toolbar option, users are somewhat divided. Maybe both features should be supported.

The same division happens with attached connections when removing an element, to leave the existing connections or remove them. Some users say keep others remove, maybe provide a choice to the user. This also happens for the borders and their elements.

One thing noticed during the tests was that users reused the connections left from elements that where deleted. They just scratched the unwanted end and draw from the middle of the remaining connection to the destination element. So it should be possible to reuse abandoned connections and attach them to other elements. Also by moving an element it near the end of an open connection, it should connect automatically.

The majority of the users used a thicker line for highlighting an element in the sketch, but some still consider possibilities adding color to the background, making the text bold and underlining the name of the elements. Of course those are features that the tool has to support in order to provide the users the possibility of highlighting elements in that way. But they do not have to be separate features in a sketching tool, the user simply wants to highlight an element, so he should have to control all of those aspects individually each time he intends to highlight an element.

For adding more complex shapes like the actor, when asked about it users preferred picking from a tool bar or from some kind panel than draw them by hand. Probably because they would find fatiguing to draw those complex shapes each time. However for more simple elements such as the activity and task widgets, users preferred drawing them. That is because they find it more quickly that having to pick form a toolbar or a toolbox.

When adding multiple elements to a sketch, users first write their names and then draw the contours. This was observed with most users, and is to avoid the name overflow situation mentioned above. In the diagraming tool they first add and roughly position all the elements and then write their names. Only then they add the connection to them. This situation in the diagraming tool happened because the users were copying it from a paper sketched diagram, so they already knew how many elements they need, their positions and names.

To name the elements users tend to pick the virtual keyboard and not the pen even though they do not discard it. This is probably because they do not have prior experience with tablet computers and handwriting recognition, so they are being resistant to change input modalities and prefer to stick with what they “know”. They actually are familiar with the physical keyboard so they think they know the virtual keyboard but probably ignore that it can be up to 4 times as slower as the physical keyboard.

Users did not found important to scratch an element to call the attention that it was not supposed to be there any more, they just scratched it because they needed to get rid of it. This is interesting as they did not find that need because actually it really was not there. As the user was not in a collaborative environment such as a meeting and the sketch would not be used by another colleague, it was just for tests purposes, they did not find a reason to transmit that information to the next person reading the diagram. The feeling was that as the element was already scratched it was no longer important; hence no further information about the discarded element was added.

Users did not understand the border concept at first, because the models being sketched are kind of a novel diagram type and not all users where familiar with it. But eventually they thought it and used it in the sketch. Obviously they added the border after sketching the elements that where going in it. So the system should recognize that elements belong to a border when one is drawn around them.

3.4 Requirements

From the user studies and the results acquired a few aspects could be passed into the requirements:

- When a name is added, if the name is too big to fit inside the shape, then the shape should grow.
- The diagram area should be “infinite”, that is grown as the diagram gets bigger or the user navigates the diagram to find more space.
- Should be possible to reuse loose connections and attach them to other elements.
- An open connection should be automatically attached if an element is placed near it.
- The system should have a highlighting feature that does not require the user to change all the visual aspects.
- Simple elements should be added by sketching them; elements that resemble primitive shapes such as the rectangle, circle or line.
- More complex shapes should be added via a toolbox or toolbar.
- Consecutive multiple element addition (same element) should be supported by the system.
- The system should recognize that elements belong to a border when one is drawn around them.
- When a connection is sketched from one element to another the system should recognize it as a connection and attach it to the two elements.
- The sketch should be represented as a formal model so it can be reused on other tools.

3.4.1 User Interface

After reviewing the literature several requirements and user needs have been elicited for User Interface Design Sketching Tools (UIDST). They are presented in this section and a list of requirements is composed.

From Plimmer and Freeman’s toolkit approach (Plimmer & Freeman, 2007), some observations related to the UI of a UIDST:

- For sketching applications there are two approaches to the user interface: a single large view or a two-view interface.
    - In a single view a “virtual page can be very large, yet display space is finite, so large pages require support such as zooming or navigation aids like radar windows”. These observations support the requirement mentioned above about the diagram area being “infinite”. (Plimmer & Freeman, 2007)
    - “Two-view interfaces typically provide a place where a collection of sketch pages can be displayed and associations established between the pages”, it is referred to as storyboard. “It suggests a linear arrangement of the sketches”. It is not an objective of this project to support storyboards, but a similar functionality could be implemented to be able to relate objects that are present in several diagrams. (Plimmer & Freeman, 2007)
- “Sketch pages provide a place where users can draw and write with a pen much as they would on paper, with support for usual computer editing such as cut, copy, paste and undo”. This type of interface, sketch pages should be the focus of the project because is in here that the sketch itself is created. (Plimmer & Freeman, 2007)
- “Backgrounds such as grids and lines are likely to be useful”. “The position of background elements may be of interest to the recognition engine for particular diagram types”. In this particular case the background is irrelevant as a point of reference for the objects;

however it could be used to delimit the diagramming area. Before it was mentioned that it would have an "infinite" diagramming area, but that is obviously impossible, as the computer resources are limited, and that would bring other concerns, for instance user spatial awareness in the diagram and performance of the application. (Plimmer & Freeman, 2007)

Kieffer et al. (Kieffer, Coyette, & Vanderdonckt, 2010) have come up with a list of requirements for UIDSTs, we can reuse that list and create a list for sketch based CASE tools:

- "Naturalness: It is necessary that the" elements "being sketched are as natural as possible first. Then, the visual similarity should be considered, to easily differentiate the various representations."
- "Non-obtrusion: It is necessary for the system supporting the sketching to be the less obtrusive as possible so as to avoid disturbing the designer during the prototyping phase."
- "Continuity: The system supporting the sketching should support the drawing continuously whatever the nature of the object" being sketched (for instance a diagram element, a connection, a text or notes).
- "Recovery: The effort provided for the sketch should be reused in the next step" of the development process. The models produced should use as much of the sketch as possible.

3.4.2 Interaction Techniques

Existing experience is that when functional gestures work they are excellent, but when they fail users get very frustrated (Plimmer & Freeman, 2007). So any gesture that is incorporated into the prototype should work properly 95% of the time or should be removed.

One thing that was thought of being a guideline for the conceptualizations of the interactions was to keep the more complex and precise gestures to the pen and the simpler and more direct manipulations to the fingers. So for instance moving or resizing an element would be done using the fingers, since those are direct manipulations of elements. And highlighting an element or adding a border around elements would be done using the pen.

From the sketching CASE tools analyzed we can see that there are two types of sketching recognition techniques. One is immediate "real-time" interpretation which recognizes the strokes as the users sketch them. The second one interprets the sketch when the user finishes it. But as seen on the tools users prefer the immediate interpretation mode, and as such that should be the mode to implement on the prototype.

4 Concept

This chapter focuses on the conceptualization of the system; as such the initial concepts which were sketched in paper will be presented first. To be able to refine the concepts a revision of the possible affordances that touch technology could bring was performed. Then from that, a mapping list of actions and corresponding affordances was devised and with that mapping was possible to describe the tasks users would do with the prototype. Finally to check the consistency of the affordances mappings a storyboard was created.

4.1 Concepts

These are the initial concepts done to start imagining what would be the interaction with the prototype. As it happens with other areas, the best way to start doing those concepts was though paper sketches. As previously mentioned they allow us to have as much freedom as possible, not constraining creativity and are a quick way to go through several concepts. The concepts are listed in Appendix 2

4.2 Affordances

NUI and the technology that makes it possible bring new affordances that were not available in more traditional interfaces. So in this section we will explore some of those affordances and compare them a bit with the affordances of other interface types.

But what are these affordances? Norman (Norman, 1988) explains that “the term affordance refers to the perceived and actual properties of the thing, primarily those fundamental properties that determine just how the thing could possibly be used.” They give clues of how thing work. “Plates are for pushing. Knobs are for turning. Slots are for inserting things into.” In this case Touch-Screens are for touching, rubbing, and sliding, holding and dragging the fingers. And a dual digitizer that can detect a pen is for writing, sketching drawing, pointing and erasing with the pen.

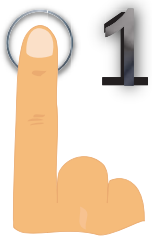
A list of essential affordances that the multi-touch technology together with the pen provides was created and can be seen in Appendix 2. This list of affordances is about what the user can do with his fingers and the pen on the touch screen, they are primarily about what the technology supports and not about the prototype’s interface affordances. But the great thing about affordances is, that when they “are taken advantage of, the user knows what to do just by looking: no picture, label, or instruction needed." (Norman, 1988) So the affordances of the prototype’s interface are also very much important.

This particular revision was made to have a greater understanding of what were the possibilities with multi-touch and to be able to refine the initial concepts.

So the main things that can easily be seen from this list are that the dual digitizer allows interacting by touching with the fingers and also using the pen, as well supporting multiple touch-points. But the affordances in the list can be diminished depending on the limitations of individual screens.

Next are three of those possible interaction affordances that both the fingers and the pen provide:

Table 6: Interaction affordances example – One finger tap, Pinch and One finger hold and pen draw.

Icon	Description
	**One finger tap**: this is for NUI what the mouse click is for traditional GUIs. It is the simplest of the affordances, but depending on the touch-screen it can have at least twice as more degrees of freedom (DOF) than the mouse click. The click has two DOFs its position (x, y) and the tap with the finger could have five, position (x, y), size and pressure, orientation.
	**Pinch**: Because it is a multi-touch is affords for more than one finger, so standard multi-touch manipulations are supported such as this one. This is good because it takes advantage of the user's abilities and skills with previous more simple multi-touch devices such as the multi-touch mouse pad on some laptops.
	**One finger hold and pen draw**: A great advantage of the dual digitizer on a touch screen is that the user can use both fingers and pen to interact with it such as holding something down with the finger and then draw with them pen over it, just like he would do with a sheet of paper a pencil. But there are some affordances that depend on the technology, the fingers and pen might not be supported at the same time, but still keeping with the example the user could first position the object with the finger and draw on it with the pen as with the sheet of paper.

To see the rest of the affordances, please refer to Appendix 2.

4.3 Actions-Affordances Mapping

From the concepts and the affordances, and combining the knowledge gained from the user studies, a preliminary list of tasks where gathered and affordances mapped to them; it can be found in the Appendix 2. This list also followed a design guideline that had surfaced of during the concept creations, but would only be applied now as not to disturb the brainstorm of concepts:

> *Use a separation principle for the interaction devices, the Pen and the Fingers. The fingers are to be used for the rougher manipulations, for instance moving, resizing, rotating and selecting elements also zooming and panning the diagram, and such. The pen would be used for the more precise interactions and gestures such as drawing elements, connections and groups, renaming an element, editing a connection, highlighting an element, etc.*

This list despite being the "finalized" concepts also acts as an inventory for the implemented interactions as it was updated throughout the project well into the implementation and user evaluations. In reality the first two stages of user implementations shaped some of the mappings seen in the list, as they are purpose was to guide de design of the interactions and the interface.

During those two initial stages of evaluation a review of the list was made to see if the mappings complied with the guideline above and as those stages where the design guide stages, the purpose of that review was to asses if any other interaction needed to be studied in the evaluations. The interactions studied on the evaluations either were deprecated if they showed negative results or kept if they were getting the evaluations approval. The results can be viewed in Appendix 2 on the observations column of the list.

For instance the affordances for adding and moving an element and also selecting an element with the pen were subject of stage 1 of user evaluations. And in stage 2 was the turn for the name edition, highlight, remove and copy elements, and again for the selection, but this time was the pen vs. the finger. These user tests can be seen in chapter 6 Validation - User Tests and Evaluation and in Appendix 3 - Evaluations.

4.4 Task Descriptions

With the affordances mapped to the actions was time to start describing the tasks users could perform with the prototype before starting the implementation of said functionalities. For that to happen a list of tasks was devised based on all the gathered information so far, form the existing tasks in the models of chapter 3 Problem Analysis and from the conclusions of the user studies and the requirements gathered. It is also based on the concepts and on action-affordances mapping, as those actions that are mapped with the affordances are the operations that compose the tasks.

Table 7: Task Inventory

Tasks	T1 - Add element T2 - Add multiple elements T3 - Connect elements T4 - Edit a connection T5 - Attach a connection T6 - Detach a connection T7 - Resize element T8 - Rotate element T9 - Move element T10 - Select element T11 - Edit proprieties of an element T12 - Name/Rename an element T13 - Remove element T14 - Make a copy of an element T15 - Add a border T16 - Add an element to a group T17 - Remove an element from a group T18 - Highlight an element T19 - Add a quick note T20 - Erase quick notes T21 - Pan the view T22 - Zoom the view

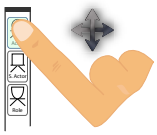
These task cards were called Task Descriptions because they are detailed versions of Task Cases (or Essential Use Cases - EUCs) which are an abstract description of tasks, they abstract from the user interface and interaction details to allow the designer to have freedom in the design and not to be constraint by any kind of technology or interaction style. However since here there was already an interaction style defined and the concepts were already including some elements of the interface these task cases became a convenient way to describe the interactions, consequently the name Task Descriptions.

Task Descriptions were not updated throughout the implementation and user tests as it would consume some more time, so some interactions and tasks might be a bit different, not implemented yet or deprecated. Check the actions-affordances mapping in Appendix 2 to have the complete inventory of implemented affordances.

The Descriptions are arranged in a card configuration, as a EUC would be except these are digital cards and not paper. On the left side are stated the user intentions, in this case with details as to how they state them and on the right side are the system responsibilities in response to those intentions.

Table 8: Task Description Cards.

T01 - Add Element

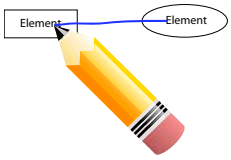
USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Indicate which element by optionally doing [ AE1 ; AE2 ]	
	2. Lock type of element in the toolbar
3. Indicate element's position in an empty space in the diagram by: If AE1 [releasing the element] If AE2 [tapping in the position]	
	4. Put element in the indicated position
	5. Unlock type of element in the toolbar
	6. Do T12 - Name/Rename an element

T02 - Add multiple elements

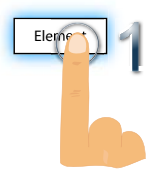
USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Indicate which element by performing  AE3	
	2. Lock type of element in the toolbar
Repeat as needed <	
3. Indicate element's position in an empty	

space in the diagram by tapping in the diagram	
	4. Put element in the indicated position >
Optionally [	
5. Perform ES2 	
	6. Unlock type of element]

T03 - Connect elements

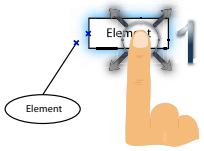
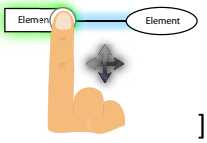
USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Indicate which two elements are to connect by performing CN1 	
	2. Offer connection types on a popup window
3. Choose connection type by tapping on it	
	4. Create a new connection between the two elements
5. Optionally [Do T11 - Edit proprieties of an element]	

T04 - Edit a connection

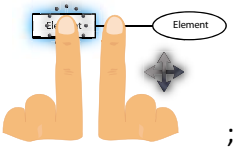
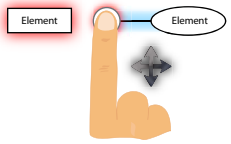
USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Indicate the connection by performing SL1 	
Repeat as needed <	
2. Manipulate connection or/and connection's handles	
	3. Save current connection >
4. Preform ES2 	
	5. Deselect connection

T05 - Attach a connection

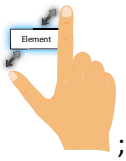
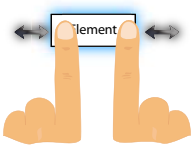
USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Indicate the element/connection to	

connect by performing optionally [CN2  ; CN3 ]	
	2. Connect the element and the connection
Optionally [	
3. Preform ES2 	
	4. Deselect connection/element]

T06 - Detach a connection

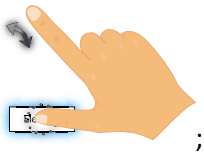
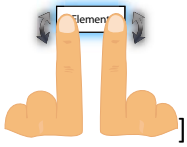
USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Indicate the connection by performing optionally [DC1  ;  DC3]	
	2. Disconnect the element and the connection
Optionally [	
3. Preform ES2 	
	4. Deselect connection/element]

T07 - Resize element

USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Indicate element's size by performing optionally [ RZ1 ; RZ2 ]	
	2. Adjust the element's size accordingly
Optionally [	
3. Preform ES2 	

	4. Deselect element]

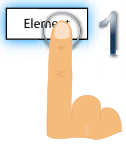
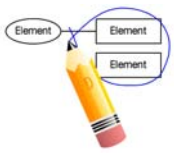
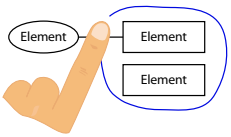
T08 - Rotate element

USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Indicate element's angle by performing optionally [ RT1 ; RT2 ]	
	2. Rotate the element accordingly
Optionally [	
3. Preform ES2 	
	4. Deselect element]

T09 - Move element

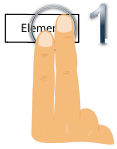
USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Indicate element's position by performing MV3 	
	2. Move the element to new position
Optionally [	
3. Preform ES2 	4. Deselect element]

T10 - Select element

USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Indicate element(s) by performing optionally [ SL1 ; SL1a  ; SL5  ; SL5a ]	
	2. Select element(s)

Optionally [Repeat as needed <	
3. Indicate additional element(s) by performing optionally [SL1; SL1a]	
	4. Add/remove element to selection >]

T11 - Edit proprieties of an element

USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Open menu by preforming BM1 	
	2. Offer contextual menu
3. Tap on edit proprieties	
	4. Provide proprieties panel for the element
Repeat as needed <	
5. Change desired proprieties	
	6. Apply proprieties to the element >
7. Close panel by optionally [performing  ES2 ; taping on close button]	
	8. Close proprieties panel

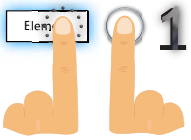
T12 - Name/Rename an element

USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Indicate the element by performing optionally [ NE1 ; NE3]	
	2. Offer |handwrite input panel  |
3. |indicate the name by writing text|	
	4. |Retain information|
Optionally [Repeat as needed <	
5. |Correct text|	
	6. |Retain information| >]
7. |Finish input|	
	8. Apply information to element and close | handwrite input panel |

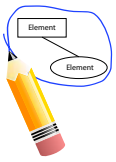
T13 - Remove element

USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Indicate element(s) to remove by  performing RM1	
	2. Remove the indicated elements

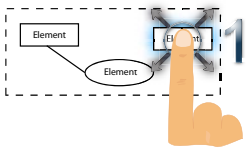
T14 - Make a copy of an element

USER INTENTIONS	SYSTEM RESPONSIBILITIES
Repeat as needed <	
1. Indicate element(s) to copy and copy's position by performing CP1 	
	2. Lock original element(s)
	3. Place a new copy in indicated position >
4. Release original element(s)	
	5. Unlock original element(s)

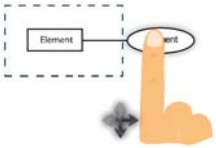
T15 - Add a border

USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Indicate elements that belong to the  border by performing BO1	
	2. Create border around the elements
3. Optionally [Perform T12 - Name an element]	

T16 - Add an element to a group

USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Indicate the element and the group by  doing BO3	
	2. Add the element to the group
	3. Adjust the border to the content

T17 - Remove an element from a group

USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Indicate the group and element by performing BO4 	
	2. Lock the border
	3. Remove the element from the group
	4. Adjust border to the content
5. Release group	
	6. Unlock border

T18 - Highlight an element

USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Indicate element(s) by performing optionally [HE2  ; HE3 ]	
	2. Change appearance of element to highlighted

T19 - Add a quick note

USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Enable quick notes by tapping the quick note toggle button on the toolbar	
	2. Enter into quick note mode
Repeat as needed <	
3. Take notes on the diagram by using the pen to write/draw them	
	4. Save the ink strokes to the quick note layer >
5. Exit quick notes by optionally [tapping the quick note button on the toolbar again; performing ES2 ]	
	6. Exit quick note mode and go back to the normal mode

T20 - Erase a quick note

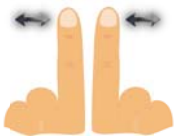
USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Enable quick notes by tapping the quick	

note toggle button on the toolbar	
	2. Enter into quick note mode
Repeat as needed <	
3. Erase notes from the diagram by using the back end of the pen to rub them	
	4. Remove the rubbed ink from the quick note layer >
5. Exit quick notes by optionally [tapping the quick note button on the toolbar  again; performing ES2]	
	6. Exit quick note mode and go back to the normal mode

T21 - Pan the view

USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Indicate direction and amount by performing optionally [ PA1 ; PA2 ]	
	2. Pan the view of the sketch accordingly

T22 - Zoom the view

USER INTENTIONS	SYSTEM RESPONSIBILITIES
1. Indicate amount by performing optionally [ ZM1 ; ZM2 ]	
	2. Zoom the view of the sketch accordingly

4.5 Storyboard

Now that the tasks are described and the interactions devised, the consistency of the affordances mappings and the interface was checked. For that purpose a storyboard was created that was based on the scenario. For this storyboard, the scenario chosen was scenario two that as mentioned before, has the most common elements of the activity models.

The full storyboard can be found in the video file Storyboard inside the CD. However this was not updated throughout the implementation and some of the interactions might be a bit different.

Figure 24: Storyboard - adding an element with the finger.

In Figure 24 we can see that the finger is being used to insert a more complex element that as mentioned before has the advantage of the user not having to draw the shape all the time and it follows NUI's direct interaction guideline. We can also see that the interface is as less obtrusive as possible having only the toolbox on the side and the toolbar on the top with the essential tools, and the main focus is on the sketching area.

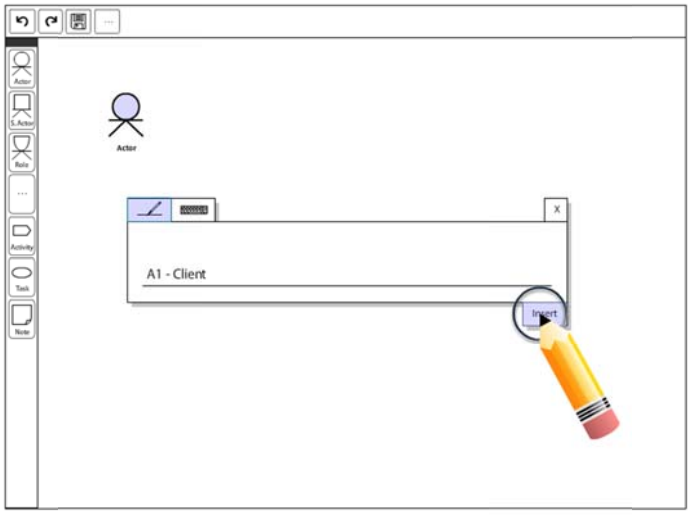


Figure 25: Storyboard - naming an element with the pen.

Next in Figure 25 the name of the actor is inserted with the pen, which is consistent with the guide line for the affordances and with NUI's instant expert guideline, as users are reusing the skill of writing with a pen.

Then in Figure 26 the pen is used to create simple shapes like the connection and the group which is consistent with the guide line for the affordances and with NUI's instant expert guideline, as users are reusing their skill for sketching. And because user only have to use the composite skill of sketching for simple shapes, it helps reduce the cognitive load, as users can be more "sloppy" with their sketches and there are not many different and complex shapes to learn.

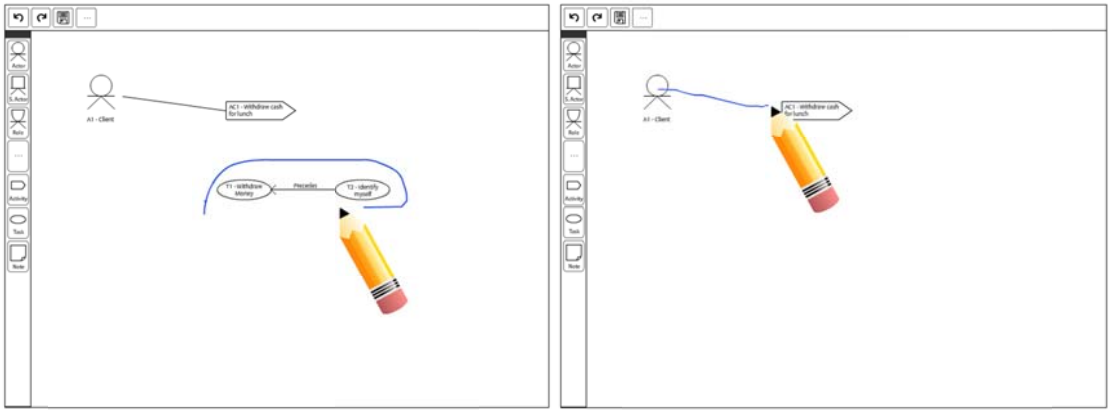


Figure 26: Storyboard - sketching a group (left) and a connection (right) with the pen.

Next in Figure 27 elements are being manipulated with the fingers which are in consistency with the guide line for the affordances and with NUI's cognitive load and direct interaction guidelines.

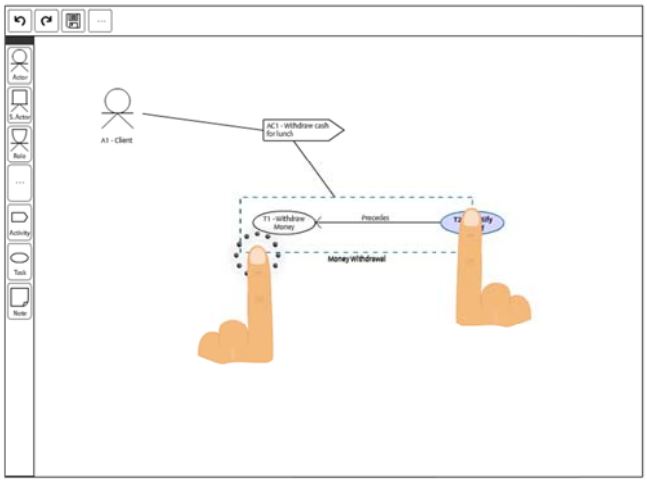


Figure 27: Storyboard - removing an element form a group.

Finally the pen is used to perform a precise gesture, in this case to highlight an element in Figure 28. It is in consistency with the affordances guideline.

As seen the interactions were consistent with the guidelines and the interface also. That meant that the conceptualization of the system was completed and that the project could continue to the implementation of the prototype. However that does not mean that the concepts cannot change, as the approach in this project followed several iterative cycles new information that come from the implementation itself and from the user tests meant that the concepts would suffer some alterations, namely the actions-affordances mapping.

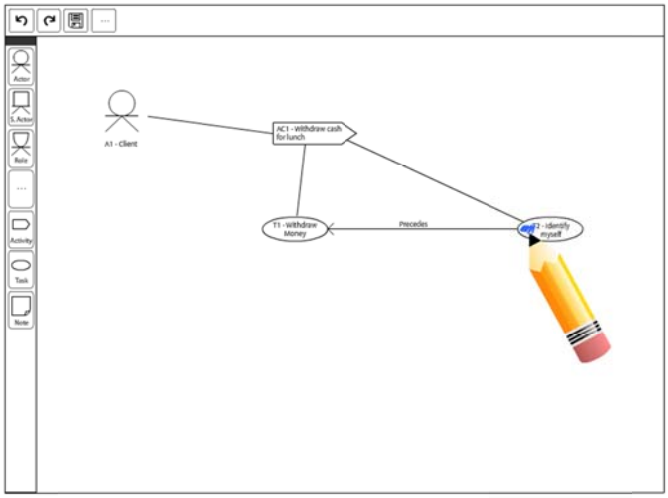


Figure 28: Storyboard - highlighting an element with a pen gesture.

5 The Prototype

In this chapter we will first see the technology used to create the prototype, both hardware and software. Then will talk a bit about the architecture and how everything works behind the UI. Then the user interface is presented with all its components, where they will be analyzed, and finally some challenges and limitations will be discussed to have a better sense of the context of some design decisions.

5.1 Hardware

The PC used in this project was a tablet-PC that obviously had to have pen as well as finger input; that was the focal point of the project. So the chosen computer was an HP TouchSmart tm2-2150ep (see Figure 18) which has a dual digitizer that can detect the fingers and the pen. The full specifications and characteristics can be found in HP website (Hewlett-Packard Development Company, L.P., n.d.).

5.1.1 Dual digitizer

The digitizer in this computer can simultaneously detect and track up to two fingers, but not fingers and pen simultaneously which can be a bit of a limitation.

The digitizer is composed by a capacitive panel layered on top of the LCD screen, and by an Electro-Magnetic Resonance (EMR) sensor behind it as seen in Figure 29. The capacitive touch panel takes the place of the protective glass seen on top of the LCD in Figure 29.

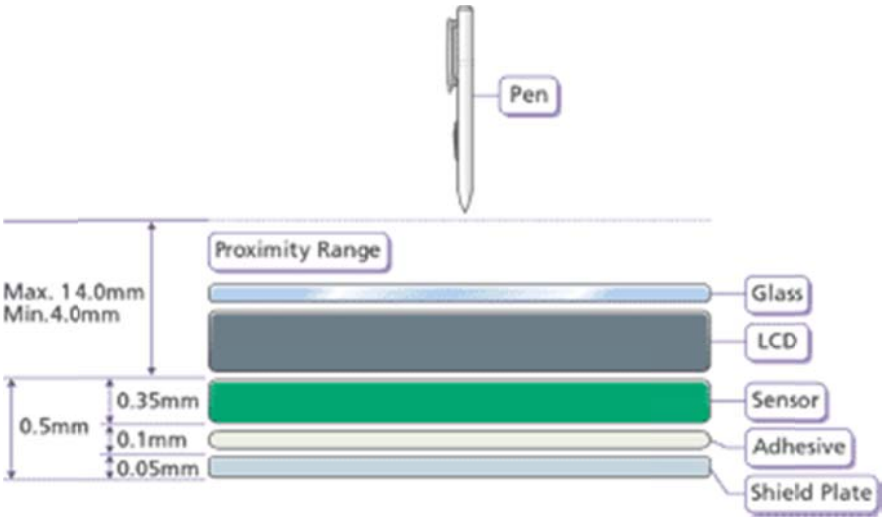


Figure 29: EMR pen detection technology. (Wacom, 2007)

By using EMR the pen can be detected at a short distance without needing to be in direct contact with the screen. That is an advantage because it means it can be used as a pointer, since the cursors on the screen can follow that movement.

In Figure 29 we can see that under the sensor there is a protective shield plate. Its job is to shield the magnetic fields that are emitted by the motherboard and other components as to not interfere with the detection of the pen.

5.1.2 Stylus (pen)

The pen that comes associated with this PC is a dual function pen with a single barrel button. It can be seen in Figure 30, where the back of the pen in on the bottom is the eraser tip, then in the middle is the barrel button, and finally on the top of the image is the main tip of the pen.

As can be seen in Figure 30 the barrel button is embedded seamlessly on the body of the pen, which makes it harder to find when operated thereby putting some limitations to the use of the button.

Figure 30: HP TouchSmart TM2 Digitizer Pen. (Hewlett-Packard Development Company, L.P, 2007)

Figure 31: EMR pen equipped with an eraser. (Wacom, 2007)

Figure 31 shows how the pen is composed and is possible to see the sensors are independent from each other. In the top part of the figure the main sensor and the barrel button which are integrated together. Then in the bottom of the figure is the eraser sensor.

Because the sensor technology is based on EMR and the sensor is embedded on the screen, the pen does not require a battery which is less of a concern to the user and it benefits the usability as the pen becomes lighter thus less fatiguing to use.

5.2 Software

A combination of libraries, framworks and toolkits was used that provided different features and capabilities to the prototype, the most important of them being the Surface Toolkit, which provides the controls and APIs on which the prototype was based.

5.2.1 Microsoft Surface

It "is a surface computing platform that responds to natural hand gestures and real world objects", and supports multi-touch of over 50 touch-points (Wikipedia Foundation, Inc., 2007). It is a combination of software and hardware however for this project only the software component was of interest. Despite, Surface can be classified as a Table Top application as its usual disposition is on a table, but the new Surface 2.0 can be mounted on a wall.

Surface was first announced in 2007 and later launched in 2008. But had been in developed since 2001 by a team formed by Bathiche, Wilson et. all. But it was not until 2003 that the first prototype was created with the nickname T1. (Wikipedia Foundation, Inc., 2007)

After being launched, the first version of the table was used by several news broadcasters and was adopted by some commercial spaces and hospitality businesses. As the world embrace touch technology and the number of partners developing Surface applications growing, Microsoft started developing the second version of the surface and it was previewed in January of 2011 at the Consumer Electronic Show (CES). (Wikipedia Foundation, Inc., 2007)

5.2.1.1 *Microsoft Surface Toolkit for Windows Touch Beta*

This was the version of Surface SDK that was used in the project. It is the software component from Microsoft Surface that has been ported to function in Windows 7, which already supports touch and stylus natively, but the Surface Toolkit "provides a collection of resources that enable" one "to create touch-enabled applications using Windows Presentation Foundation (WPF)" (Microsoft, n.d.), which can run on touch capable PCs. Those resources include Surface controls, APIs, templates and samples that allow to easier and faster create multi-touch applications and to have the advantages of the Surface touch interface on the PC.

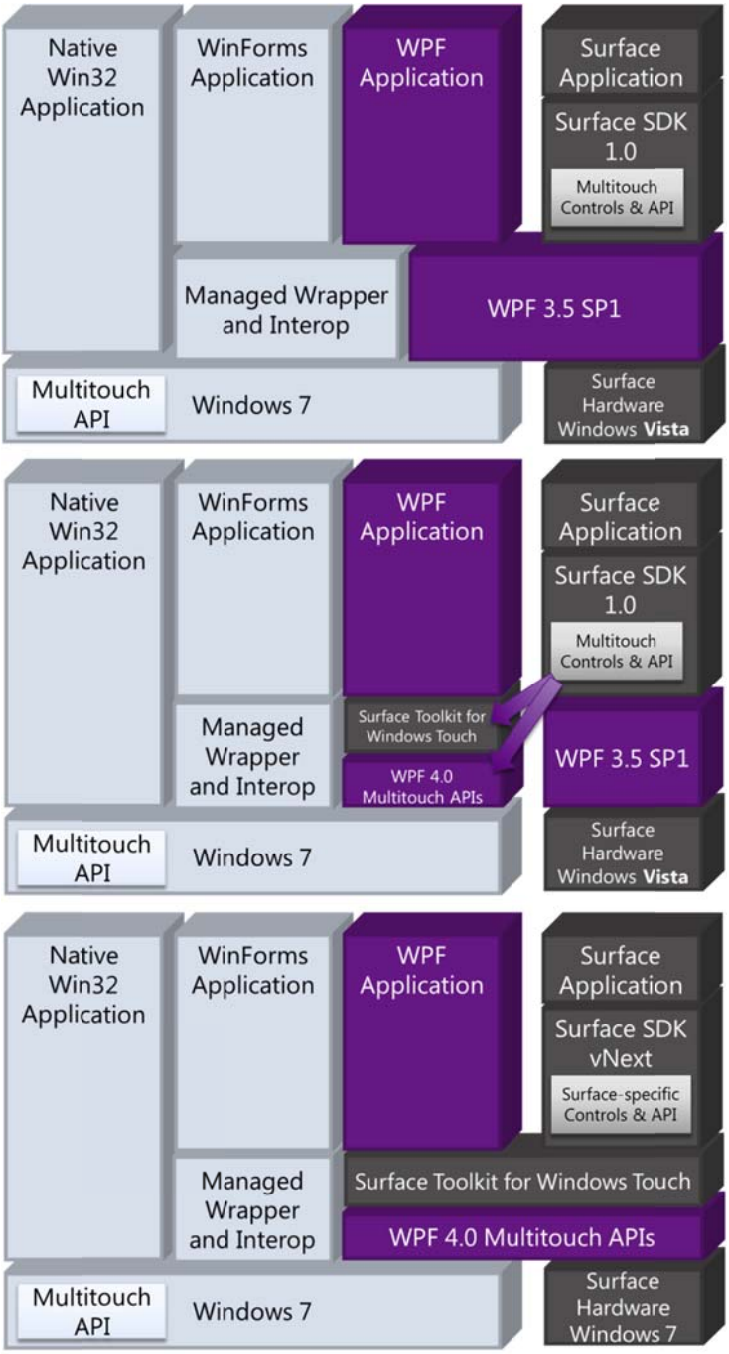


Figure 32: Touch Development in 2009 (top), in 2010 Q1 (middle) and the trend (bottom). (Feldkamp, 2009)

As can be seen at the top of Figure 32 the first version of the Surface was developed over WPF 3.5 SP1 and it was completely separate form Windows applications. Then in 2010 the toolkit was created from the Surface SDK 1.0 using the multi-touch Controls and APIs. It was paired on top of WPF 4.0 which was introduced with multi-touch APIs, and that combination allowed for WPF Applications to both support multi-touch natural interactions and to have the controls from Surface where to use those interactions.

The tendency is for the WPF 3.5 that supports surface to be replaced by that combination of the Surface Toolkit and WPF 4.0 as seen in the bottom of Figure 32. That will allow for applications to be developed on one system and run on the other.

5.2.2 Libraries and Frameworks used

As can probably be deduced from the previous section this prototype was developed on Microsoft Windows 7 and over the Microsoft .NET Framework 4.0 and WPF 4.0 using the Microsoft Surface Toolkit for Windows Touch Beta. The editors used were the Microsoft Visual Studio 2010 and Microsoft Expression Blend 4. Blend was used to design the interfaces and the animations, where Visual Studio was used to code the prototype.

Blake.NUI library was also used, which "is a collection of helpful controls, utilities, and samples useful for multi-touch and NUI development with WPF 4 and Surface". Despite being still in an "alpha status", this library already gives some fundamental functionalities such as the possibility to detect finger tap, double-tap and hold on any component in the interface. (Blake, "Blake.NUI" at CodePlex, 2010)

The Microsoft.Ink library was used to provide the handwriting and ink gesture recognition capabilities.

5.3 Architecture

In this section the structure and the composition of the prototype are presented, firstly an overview of the architecture created which will cover the most important aspects of it. Secondly, and because this is an event based architecture, the events used will be explored. Finally the components created will be presented.

5.3.1 Overview

In Figure 34 is a Class Diagram of the structure and the architecture of the prototype, it is not a complete diagram because then it would have become very large and complex. So this diagram focuses on the main structure and software mechanisms that provide the prototype with its functionality.

It is important to state at this point that the architecture is an event paradigm architecture which is based on the events from Windows and the Surface toolkit. With that in mind the analysis of the structure itself can be started.

By splitting the diagram into six parts as seen in Figure 33 and analyzing each part individually, the diagram becomes easily understood. Group 1 is concerned with the Diagram Elements and group 2 is about the Connections. Group 3 contains the interfaces that both the Diagram Elements and Connections share and which provide that common functionality to both of them. Group 4 is the Drawing Box that is where the Diagram Elements and the Connections meet. Group 5 is concerned with the Widget Panel and finally group 6 is the Main Window where all the components come together.

Group 1

Looking now with more detail into each group and their relationships. Starting with group 1, the class DiagramElement is the class that has the major functionality of the elements that compose a diagram. Its base behavior comes from the Surface's ScatterViewItem, which provides functionalities such as rotate, scale and move. Then in the class itself we can see that it has responsibilities such as managing the visual state highlighted, manipulating all the other elements that are selected when it is manipulated though the DiagramElement_ContainerManipulationDelta() and detecting when any of its properties are changed and letting others know those changes through the OnPropertyChanged() operation and the interface INotifyPropertyChanged. Additionally the Diagramelement has to know if a point or a ConnectionHandle is over itself, which is needed when an element is traveling over a DiagramGroup or a ConnectionHandle is traveling over an element, to know if that element or handle will be attached to its target when dropped, that is the element that returns true. However when is the element that is traveling it queries elements that are underneath him using the operation FindDiagramElementUnderneath() and an element will not be added to a group if it is present in its ignore list. Moreover the DiagramElement class knows how to set its visual state and to represent itself as stylus points. These last four operations are left for the subclasses to implement.

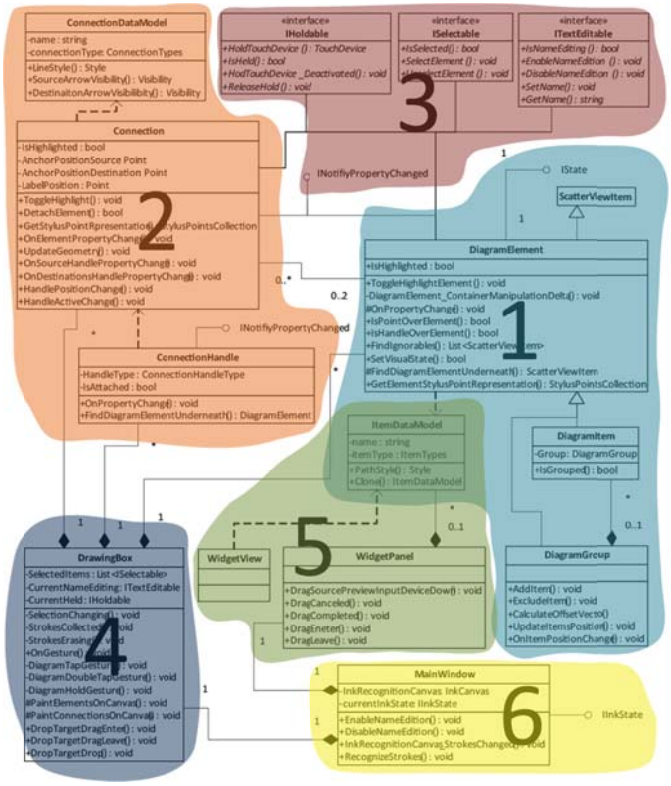


Figure 33: Class Diagram's division - Group 1: Diagram elements; Group 2: Connections; Group 3: Interfaces; Group 4: Drawing Box; Group 5: Widget Panel; Group 6: Main Window.

A DiagramGroup, besides having all the functionalities of the DiagramElements can contain items with itself; they can be added and excluded. The DiagramGroup has to calculate an offset vector for each item it contains so that when it is moved, the items can be moved with it, using UpdateItemsPosition() operation to update their position. The vectors are also calculated when an item changes its position, which the group subscribe to and is updated through the operation OnItempositionChange().

The DiagramElement has the ability to keep track of the current state of the element through the IState interface. IState is an implementation of the State software design pattern; it provides the ability to keep track of the current state and enables the element to change to another state when required. Further down in section 5.3.2 Events this state will be further explored with a focus on the events leading to changes in state.

The DiagramElement depends on a data model to store its information. In this case the ItemDataModel responds to that need by storing the name of the element and which type it is. Having this model and the DiagramElement class allows for new elements to be added more easily to the prototype. The style of the elements is stored on a style sheet in a resource dictionary, as a path shape and can be retrieved using the operation PathStyle(). Having the styles on a style sheet allows designers to work independently form developers on their own tools.

Group 2

A similar situation as with the Diagramelement's data model occurs with the connections in group 2 as there is a ConnectionDataModel, which stores the name of the connection and its type. Adding new types of connections then is easier. Additionally depending on the type of connection arrows on the ends of it can be visible or not, and it is in the data model this information can be retrieved through the operations SourceArrowVisibility() and DestinationArrowVisibility().

The class Connection is responsible for the core functionality of a connection. As with the DiagramElement it manages the highlighted visual state of the connection as well as the other visual states. It is also responsible for detaching itself from any element when required, it does so through the operation DetachElement(). As the Connection has a compound and spread representation on the prototype it is required to keep track of all its different components and to update its geometry when the components change. For that purpose the operation OnElementPropertyChange() listens to changes in properties form the attached DiagramElements, and the operations OnSourceHandlePropertyChange() and OnDestinationHandlePropertyChange() are subscribed to the properties of the source and destination ConnecitonHandles respectively. Those two work in conjunction with the operations HandlePositionChange() and HandleActiveChange() to process the changes in those properties. Next in the Events section these event operations are explored with more detail. To simplify things a bit the line that represents the connection binds to the ConnecitonHandles using the WPF binding capabilities, and automatically changes as the handles are displaced. The advantage of this is that there is no further code needed for the line to follow the handles.

Both the classes Connection and DiagramElement have operation GetStylusPointRepresentation, these operations are about the different layers that the DrawingBox is composed with. As each layer provides different functionalities there is the need to represent the elements and connections in different layers. For that reason Connection class is not a specification of the class DiagramElement, because despite sharing a few of the characteristics, like selection, text edition, highlighting capability, they are too different in terms of function and inside structure. The layers will be explored later in section 5.3.3.2 Drawing Box, where the DrawingBox is presented in more detail

Still within group 2 we find the ConnectionHandle which has the main goal of providing direct manipulation of the connection endpoints to the user. They are a separate class because of the different layers. The handles reside in the same layer as the DiagramElements, the ScatterView, and implement the INotifyPropertyChanged so the Connection can keep track of its changes as mentioned above. The ConnectionHandle contains the information of which

type of handle it is (source or destination) and it knows if it is attached to a DiagramElement. Additionally it has the responsibility of finding other elements that are underneath it when it is traveling through the sketch, as with the DiagramElements.

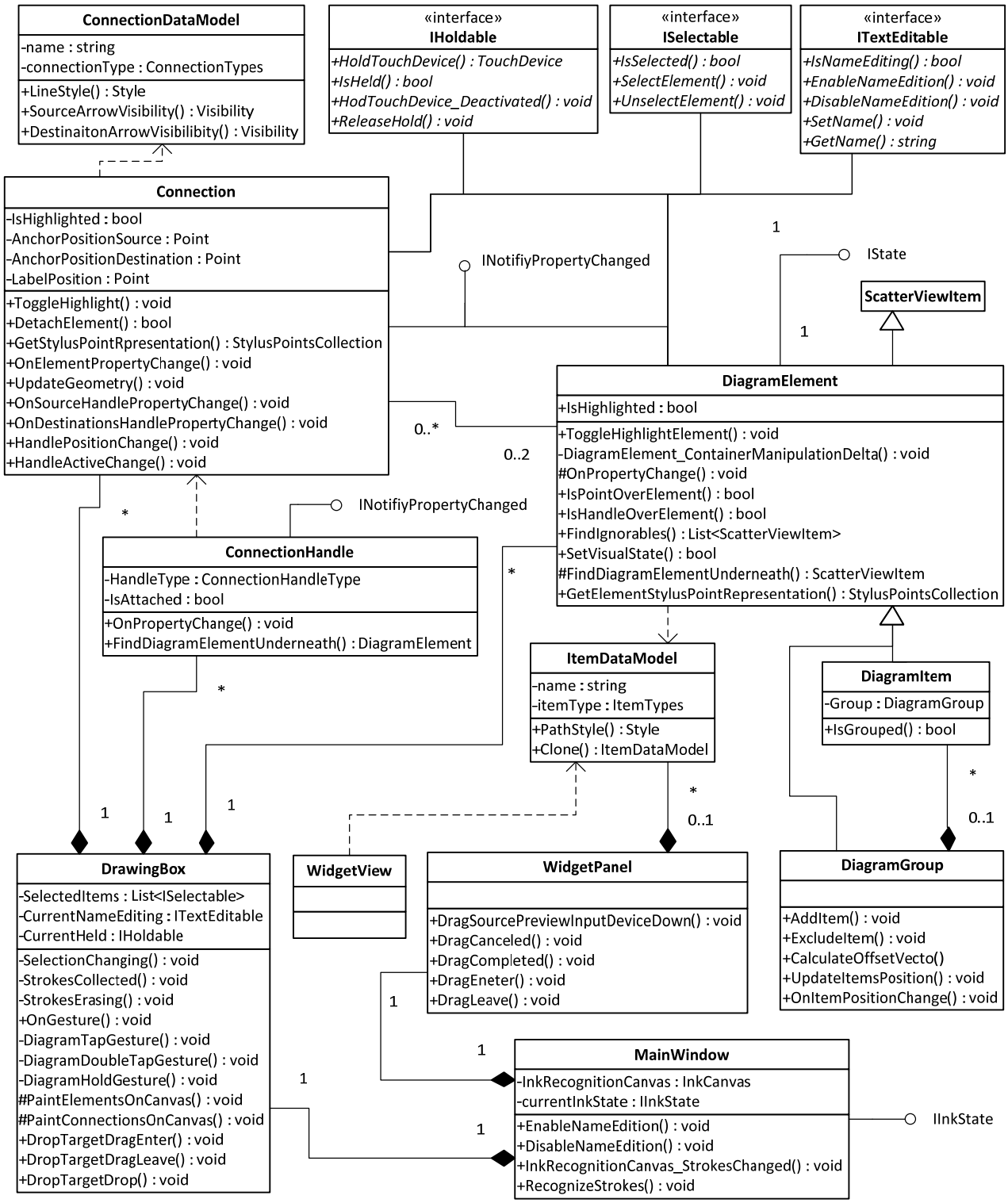


Figure 34: Class Diagram.

Group 3

On the top of Figure 33 there is group 3, which consists of the interfaces devised for the prototype. As mentioned before they provide the equivalent functionality that both the

DiagramElements and the Connections share. There is the interface IHoldable which provides the management capabilities to deal with elements and connections that are being held by the user; this interface works in conjunction with NUI library (Blake, "Blake.NUI" at CodePlex, 2010), which provides the actual hold capability. Then the ISelectable interface provides the selection functionality, and finally the ITextEditable interface which provides the functionality of editing text from a Connection or a DiagramElement. The INotifyPropertyChanged interface as mentioned before provides to elements, connections and connection handles, the capability to report any changes to their properties to any object that has subscribe to them.

Group 4

In group 4 is where group 1, 2 and 3 come together. They are aggregated into the class DrawingBox which is where the main interaction happens. This class holds the ScatterView and the various Canvas and InkCanvas that support the connections, the stylus gestures and input, and the QuickNotes feature. It is also in this class that selected items, the component that is editing its name and the component that is being held by the user are registered.

The DrawingBox is the one that has to detect when elements are being selected or highlighted, and when complex and simple elements are being added or removed. This means that it has to support both pen and fingers interactions. So to select multiple element the operation SelectionChanging() detects when diagram components are being selected or deselected. The operation StrokesCollected() detects when simple elements are being added to the diagram and the operations DropTargetDragEnter(), DropTargetDragLeave() and DropTargetDrop() detect when complex elements are being added from the WidgetPanel. StrokeErasing() detects when elements are being removed from the diagram. Operation OnGesture captures the gestures created by the pen such as highlighting elements and selecting elements.

Operations PaintElementsOnCanvas and PaintConnectionsOnCanvas are used to represent the elements and the connections on the GesturesInkCanvas, which uses those representations in conjunction with the input form the pen and fingers to select, remove, create or highlight elements.

To directly manipulate elements using the fingers, additionally to the functionality provided by Surface's ScatterView and ScatterViewItems, the operations DiagramTapGesture(), DiagramDoubleTapGesture() and DiagramHoldGesture() are registered in the class DrawingBox. These operations are provided by the Blake.NUI library. They function here because of the disperse nature of the several components that comprise a diagram; where connections are in one layer and elements are in another. Having the taps and holds detected in the DrawingBox allows having a better control of which element will respond to that input.

Group 5

Within group 5 is the WidgetPanel which is used as a toolbox to add the more complex elements to the diagram. For that purpose it has operations that support the drag and drop interaction. The WidgetPanel also uses the ItemDataModel to store the item's data.

The WidgetView is a visual representation of the items that is used to represent the elements in the WidgetPanel and is also used as cursor during the drag and drop operations, is a slightly simpler view than the one used in DiagramElement because it does not need to support the same functionalities.

Group 6

This group is composed by the MainWindow and the IInkState. The MainWindow is where all the components come together to form the complete prototype's interface, the DrawingBox, the WidgetPanel. Additionally on the MainWindow class there is embedded another component, the InkRecognitionCanvas; is in this canvas where the handwritten text is inputted and then recognized. The operations InkRecogniotinCanvas_StrokesChanged() and RecognizeStrokes() provide that functionality of recognizing the handwriting, while the operations EnableNameEdition() and DisableNameEdition() activate and deactivate the functionality.

The IInkState interface provides the ability to keep track of the current ink state and as with IState from class DrawingElement, it is an implementation of the State software design pattern. The ink state tracks all the interactions with the pen that are related with ink insertion, selection or removal. In addition the ink state works cooperation with controls on the toolbar from the MainWindow, which allows for users to select elements with a lasso using the fingers. Next the ink states are explored in more detail.

5.3.2 Events

As mentioned earlier the architecture is based on the event paradigm, therefore here will be presented some of the more important events that compose the architecture and how they are used throughout.

5.3.2.1 *Touch*

Form Windows 7 the Raw Touch API there are a "small number of events, properties, and methods that are integrated into" WPF. These provide access to all the possible touch interaction bits. In Figure 35 is an illustration of those events in action, the figure shows the events and the order in which they occur, when a finger is swiped over the interface.

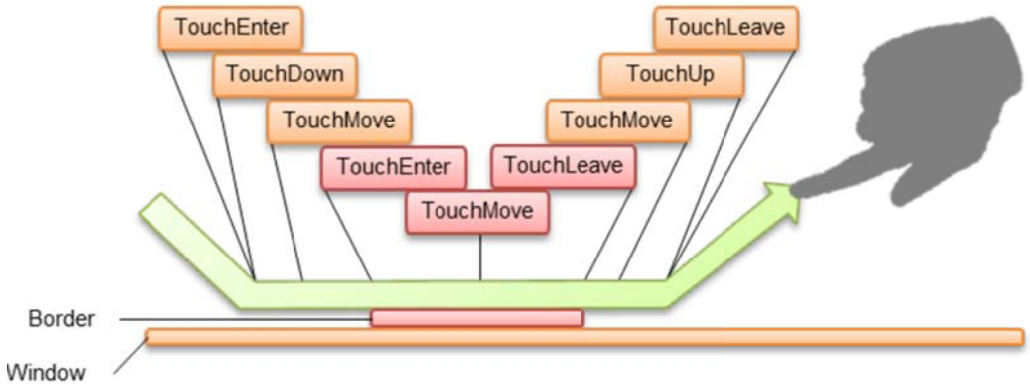


Figure 35: Sequence of touch events from Windows 7 Raw Touch API; starting from the left with TouchEnter and ending with TouchLeave on the right. (Blake, Natual User Interfaces in .NET, 2011)

We can see that there are repeated events as there are two interface components on which the finger directly interacts, a Window and inside it a Border. So when the finger firstly touches the screen the first even fired is the TouchEnter, then TouchDown. Afterwards when the finger moves is the TouchMove and then when the finger starts going over the Border the TouchEnter is fired again, but this time it has a different sender which is the Border. Then the finger moves and leaves the Border fiering the TouchLeave. Then it moves again over the Window, firing TouchMove again. When the user finally lifts the finger the event TouchUp is fired and finally TouchLeave is the last event registered.

In the next table there is a description of the events seen on Figure 35 and a mouse equivalent is provided also which helps to understand the resemblance between the mouse and touch.

Table 9: Raw Touch API UIElement class' touch events and mouse equivalents. (Blake, Natual User Interfaces in .NET, 2011)

Touch Event	Mouse Equivalent	Description
TouchDown	MouseDown	Occurs when a finger first contacts the touchscreen over an element
PreviewTouchDown	PreviewMouseDown	TouchDown event routed from root element to source element
TouchMove	MouseMove	Occurs when a down finger moves over an element
PreviewTouchMove	PreviewMouseMove	TouchMove event routed from root element to source element
TouchUp	MouseUp	Occurs when a down finger is lifted from the touchscreen over an element
PreviewTouchUp	PreviewMouseUp	TouchUp event routed from root element to source element
TouchEnter	MouseEnter	Occurs when a down finger enters the boundary of the element as well as just before a TouchDown event
TouchLeave	MouseLeave	Occurs when a down finger leaves the boundary of the element as well as just after a TouchLeave event
GotTouchCapture	GotMouseCapture	Occurs when a touch is captured to an element
LostTouchCapture	LostMouseCapture	Occurs when a captured touch is released from an element

This resemblance and parallelism does not mean that touch is merely a mouse that works with the finger. It is very much more than that. The events that support touch are full of information that the mouse simply cannot provide. As previously mentioned, besides the position the touch amongst other parameters, can report pressure, size and orientation.

There is a richness of information that makes the interaction so much more natural and complete. Yet, still there is the aspect of multiple touch points vs. single pointer.

5.3.2.2 *Diagram Element States*

As said in the Overview a state pattern was implemented to keep track of the DiagramElement's state. In Figure 36 is a state chart that explains these states.

On top of the figure we can see that the DiagramElement's initial sate is the DefaultState. Then if the user double taps the element or does a gesture with the pen that is meant to start editing the name, the state changes to IsNameEditingState. Then if a double tap event is detected again over the element or if the Ink Recognizer from the MainWindow stops editing the name, because the user has finished inputting the name, the state is then changed back to the DefaultState.

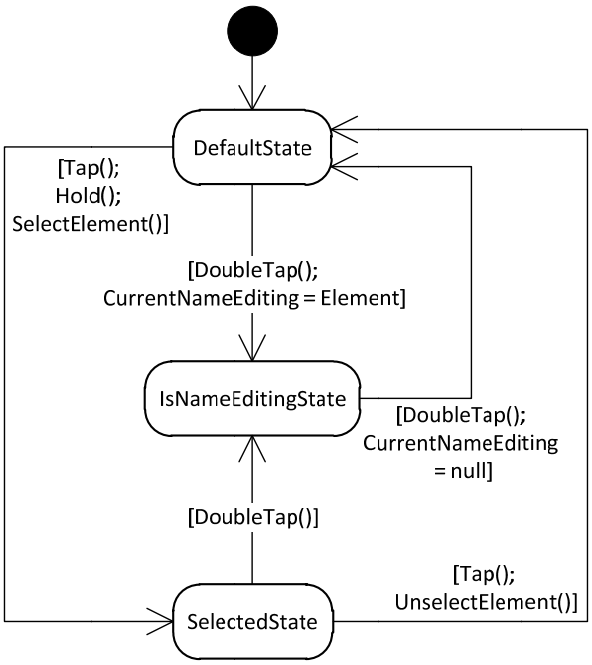


Figure 36: Diagram Element's state diagram.

When in default state if a tap or hold event is detected over the element the state is changed to SelectedState, which also happens if the user does a multiple element selection with the lasso. In the last case the operation SelectElement() is used to switch to SelectedState. If a double tap event is detected then the state changes to IsNameEditingstate, but if a tap is detected over the element or on whitespace the elements is unselected and the current sate becomes the DefaultState.

5.3.2.3 *Ink States*

As mentioned in the previous subsection a state pattern was implemented for the ink interactions. The IInkState interface tracks all the interactions with the pen and the controls on MainWindow's toolbar, these controls can be seen in section 5.4 The Interface/Design.

About the toolbar controls, if for instance from any other state the selection toolbar control is used, and the button click event is fired, the state of the ink is then switched to SelectState. The same analogy can be applied to the other states. In any active state (except NoInkState), if the toolbar control for that sate is used the state is deactivated and falls back to NoInkState. Then these controls can be viewed as toggles that activate and deactivate the states.

Figure 37 presents a state chart of the implementation and we can see the first state is the NoInkState, then if an event concerning the pen or the ink controls is detected the state is changed accordingly. When the stylus is in range and is not inverted (the writing tip facing the screen) or the ink button on the toolbar is pressed, the state is changed to InkState. If the stylus is in range but is inverted (the erasing tip facing the screen) or the rubber button on the toolbar is pressed the state changes to RubberState.

When in InkState if the stylus barrel button is pressed the state changes to SelectState. Note that the same does not happen when in RubberState, because the pen is inverted there and the barrel button does not function when the pen is inverted. If the pen is moved away from the screen and the toolbar ink control has not been used to activate the state it goes back to NoInkState, if the ink control has been used then it can be used again to deactivate ink

and to go back to NoInkState again. In in InkState the user presses the barrel button on the stylus the state is switched to SelectState.

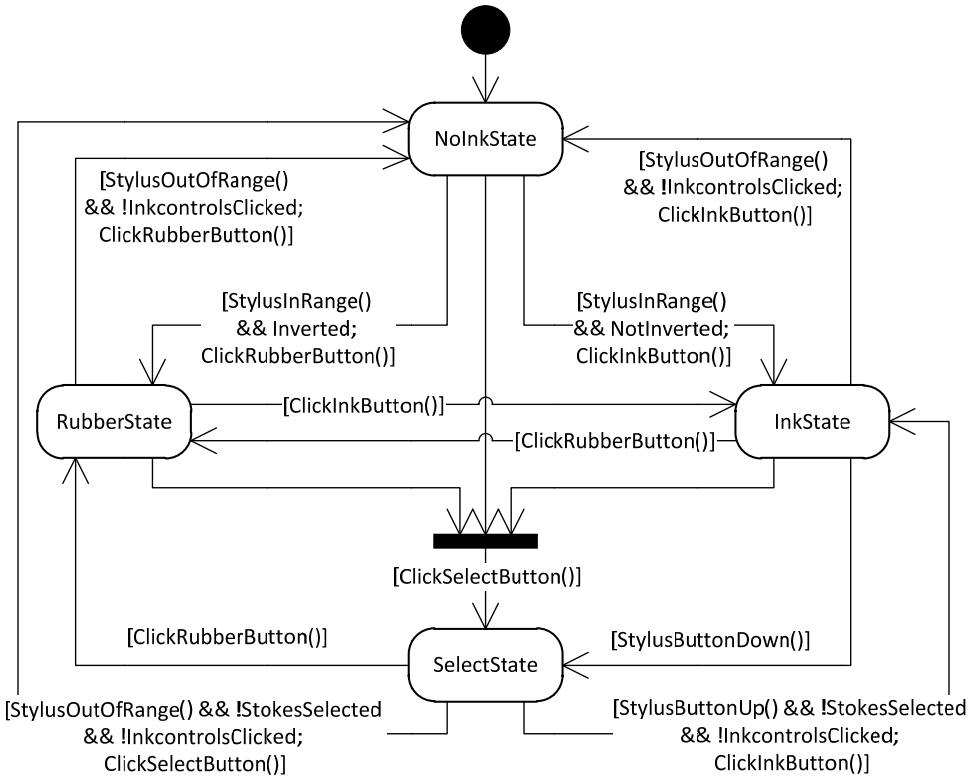


Figure 37: Ink state chart.

Then when in SelectState, if the barrel button is released and there were no strokes selected and the toolbar control has not been used to activate the state, it falls back to InkState, otherwise it maintains selected. It will also change to InkState if the toolbar ink control is used. If the user backs away the pen and there were no strokes selected and the toolbar control has not been used to activate the state the state falls back to NoInkState. The selection toolbar control can also be used to deactivate the state.

In RubberState if the stylus goes out of range and the toolbar controls are not used, then the state switches to NoInkState. The rubber control can also be used to deactivate the state.

5.3.2.4 *Connection Geometry update*

As stated before the connections are divided through the layers on the DrawingBox, and when one of its components changes the connection has to change with it. To detect those changes the connections rely on the events to transmit which were the changes that occurred, especially on property change event type that indicate the object and property that has changed.

Figure 38 presents a DFD that explains how those changes are processed, the objects involved and the events and data used.

On top of Figure 38 are the objects that are involved in this process. Starting with the Connection Line we can see that it binds to the center of the Connection Handles, as it was mentioned before it simplifies the geometry update process a bit because there is no need to do update the line “by hand” since the binding does that automatically.

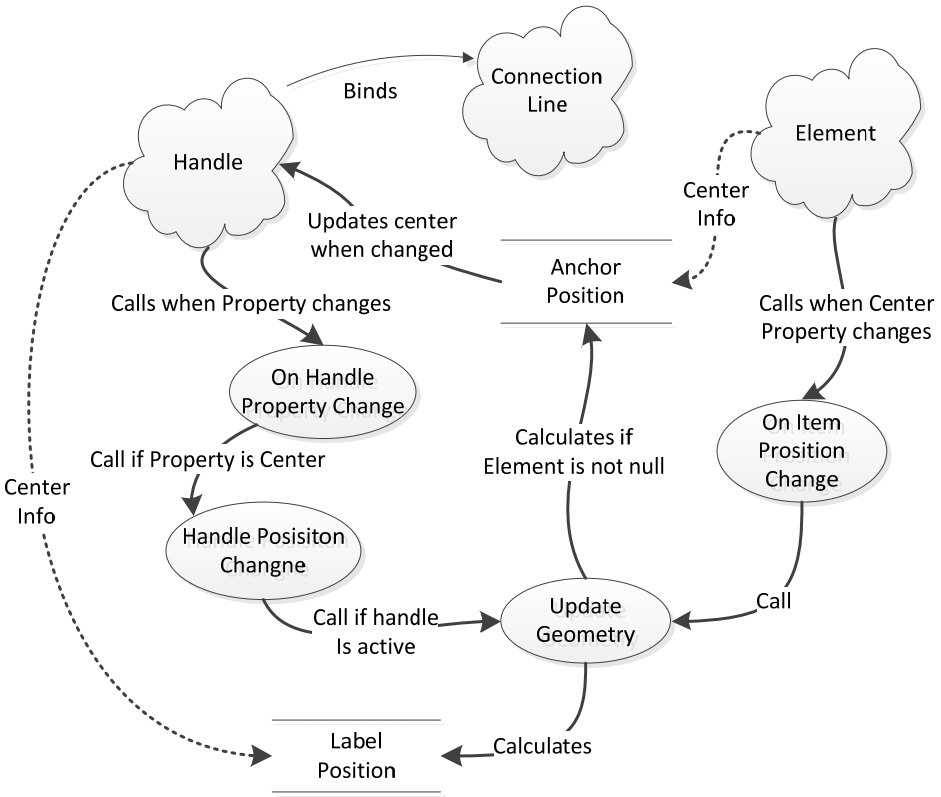


Figure 38: Connections geometry update's Data Flow Diagram (DFD).

When the Connection Handle is moved by the user, it fires a property changed event that is captured by the Connection's On Handle Property Change event handler, if the property is the center then the Handle Position Change operation is called. The Update Geometry method is only called if the handle is being manipulated by the user, if it is active. If so the Label Position is calculated using the Center information from the source and destination connection handles, also if the connection is attached to Diagram Elements, the Anchor Points for the handles are calculated.

The Anchor Points are points that float near the Diagram Elements and they are where the Connection Handles are attached. Furthermore if the user is interacting with one handle, the Anchor Point's calculations only occur to the other end's handle if it is attached to an element. After the calculations for the point are done the center of the Connection Handle is updated, stopping the cycle when the handle is check to see if the user is interacting with it.

If the component changing is the attached Diagram Element, then a property change event is fired and captured by the connection's event handler On Item Position Change, which calls Update Geometry. It calculates the Label Position and the Anchor Position using the Diagram Element's Center info for the connection end that the element is being moved, for the other end only calculates the Anchor if there is an element attached. Finally it updates the Connection Handle center to the calculated Anchor.

5.3.3 Components

Three components were created that could be reused on other projects; they are the WidgetPanel, the DrawingBox and the MainWindow. Next is a description of them.

5.3.3.1 *Widget panel*

In the WidgetPanel elements are represented using the ItemDataModel as their data model and the WidgetView as their style, which means the WidgetPanel can be reused for other toolboxes of other languages or even on other projects. To reuse the WidgetPanel one has to simply switch the data model with the one needed and do the same with the style of the elements.

The Drag and Drop functionalities are able to deal with other data models and styles and so as the WidgetPanel is reused there is no need to do any more changes to the panel other than the ones referred above.

5.3.3.2 *Drawing Box*

The Drawing Box is where all the diagram elements are aggregated. However the different functionalities are divided through several layers. In Figure 39 the order of those layers are present. Starting from the back he ZoomCanvas (Blake, "Blake.NUI" at CodePlex, 2010) is the layer that provides the zooming capability of the diagram and it contains all the other layers.

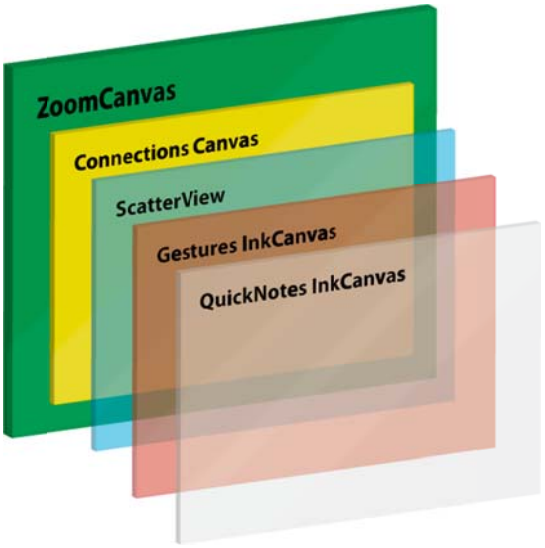


Figure 39: Drawing Box's canvasses layer composition.

Then the Connections Canvas is where the main part of the connection is represented, that is the line that represents the relationship between two elements. The ConnecitonHandles are on the next layer, the ScatterView. This is the layer form the Surface toolkit that provides the natural manipulations of elements; those are the scale, rotate and move. So is in that layer that the DiagramElements reside.

The next layer is Gestures InkCanvas which is where the pen interactions are detected. So here is where the creation of elements by sketch, the removal of elements and the selection of elements using the pen are detected. For that to happen the elements have to be represented into this layer as stylus ink so it can be selected and removed with the stylus, however it can be represented as shapes and points which can then be checked for bound intersections or containments in the case of the creation of connections and groups. Next are some examples.

For instance the creation of connections as represented in Figure 40, are recognized by detecting the initial and final points of the stroke the user created, represented in the figure as the corresponding blue and red circles. By representing the elements in Gestures InkCanvas

Figure 40: Representation of the elements on the GesturesInkCanvas for the detection of the connection gesture and.

with boxes that correspond to the bounds of the elements, which are the blue boxes in the figure, the prototype then calculates if the points are within the bounds of the boxes and creates the connection if so.

Next in Figure 41 there is the creation of a group. The stroke the user does to group the elements is passed to a polygon that is represented by the semitransparent red polygon in the figure. Creating groups uses the ray tracing, which uses the points form the center of the elements, represented in yellow circles in the figure, to see if they are within the polygon.

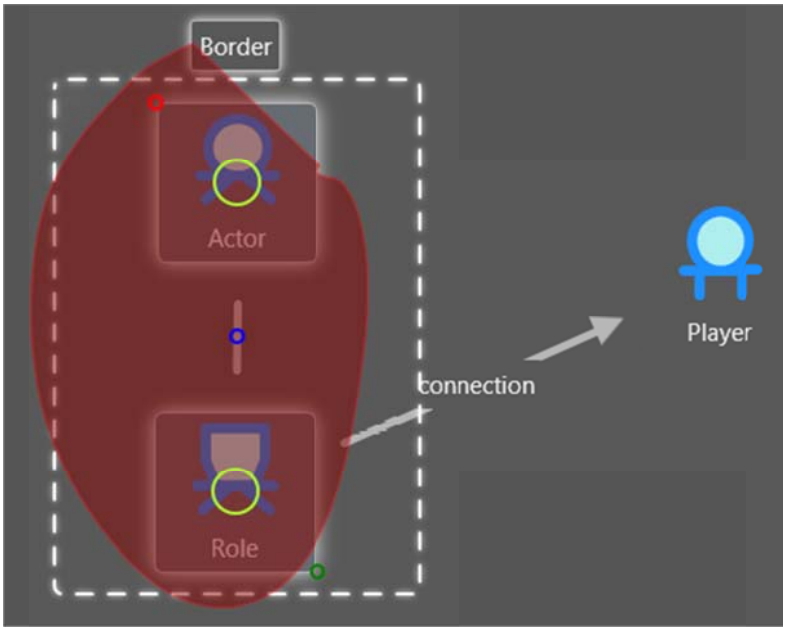


Figure 41: Representation of the elements and connections on the GesturesInkCanvas for the creation of a group using a pen gesture.

If they are then the bounds of the group are calculated and the group is created with a minimal height and width. In Figure 41 the reference points are represented by the smaller circles: the top left red circle represents the top left corner of the bounds, the middle center blue circle the center point which is used as the position for the group itself and finally the lower right green point that represents the bottom right corner of the bounds. Then the group is added to the position calculated and with the proper dimensions plus a small margin.

In Figure 42 the elements and connections are represented in the Gestures InkCanvas so they can be removed. In the case of removing they are represented as more than points to give the user a better chance to hit the intended element. The groups are not crossed so the user can remove only the group or the elements inside it.

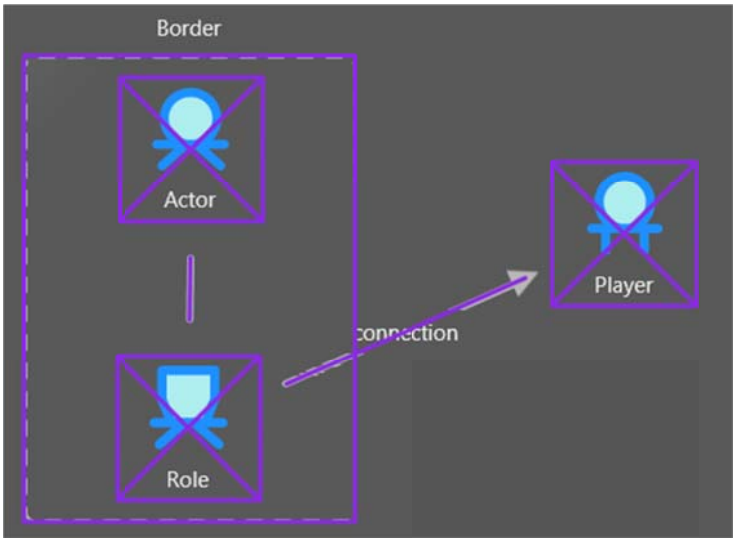


Figure 42: Representation of the elements and the connections on the GesturesInkCanvas for removing.

Selection works in a similar way as the removal, the elements are represented as ink on the Gestures InkCanvas as in Figure 43.

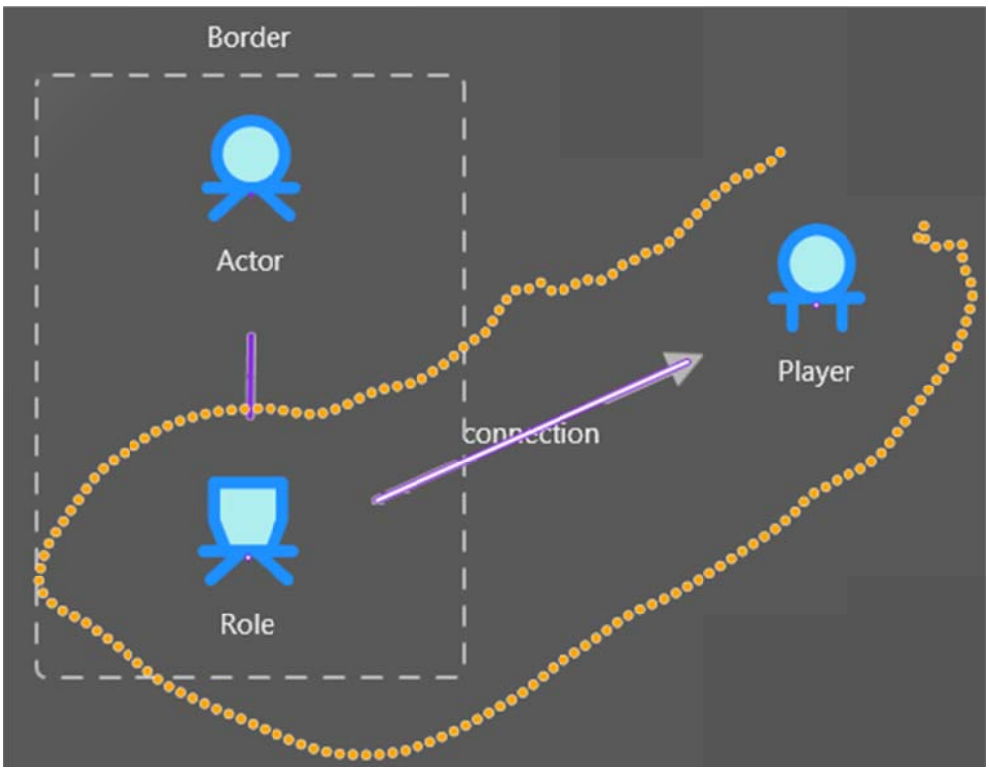


Figure 43: Representation of elements and connections on the GesturesInkCanvas and selection.

The elements are represented only by their center points and the connections by a stroke from one end to the other. Then normal ink selection can be used to select the elements.

The highlighting functionality, works in a similar fission as user produces the gesture and the system represents the elements in the Gestures InkCanvas, then the system has to check if the gestures was with the bound of the element, or if the element is with the bound of the gesture. The user only has to perform a gesture for the element to get highlighted and not change every aspect of it (color, contour, text).

Other functionalities are implemented in the same manner as the examples presented here.

5.3.3.3 *MainWindow*

The MainWindow is where all the components are gathered, but it has a very important component imbedded in itself which is the InkRecognition and the ink controls. The ink recognizer is not a real component, because it is embedded in the main window. However just by separating it we could create a component that can be reused in any other interface and that when required collects the strokes and recognizes the handwritten text using the Windows Ink Analizer.

5.4 The Interface/Design

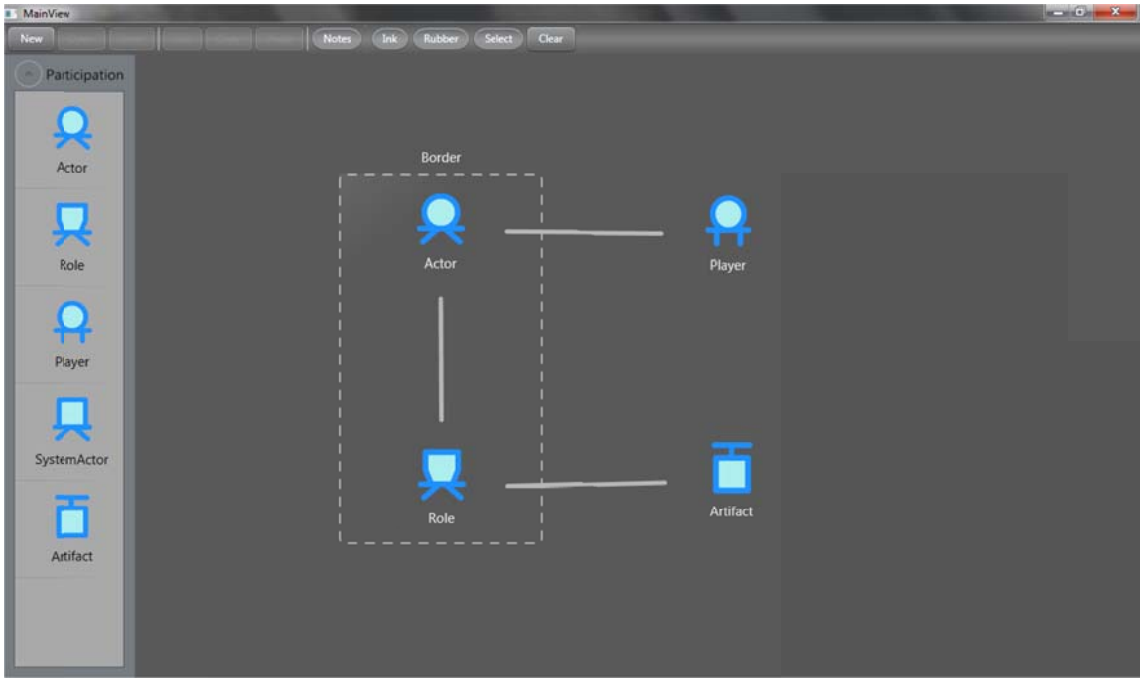


Figure 44: User interface, main window.

In Figure 44 we can see the main interface. At the left there is the WidgetPanel, at the right center is the sketching area and at the top the essential controls. This design is in compliance with the requirement that the interface should not be obtrusive. As seen the sketching area is only slightly occupied by the WidgetPanel and the Toolbar, being that the majority of the space is attributed to the sketching area.

The Toolbar make users instant experts, as it supports legacy users. The toolbars can also be considered containers and the buttons objects that are manipulated when the user presses them. By adding animations to teach the users about the possible gestures and manipulations of the various objects, the progressive learning would be supporting. For instance in the copy/paste buttons, the ink buttons, animations showing the interactions with the pen or the fingers, could be added that paly when the user uses the button.

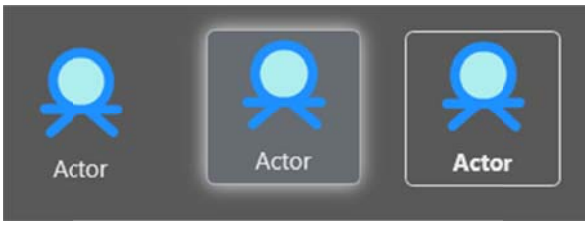


Figure 45: Possible states for the elements: default on the left, selected on the center and highlighted on the right.

The several visual states of the elements are presented in Figure 45 and in Figure 46 the connection is selected, which in that state shows the handles on the end points. The handles can be picked up by the user, which will then grow as seen in Figure 47. Also in the figure is presented the visual state change that the handle suffers when it can be attached to an element, its glow changes to green as does the elements. Also in the figure there are the changes that happen when the connection is being detached from an element, the handle becomes red.

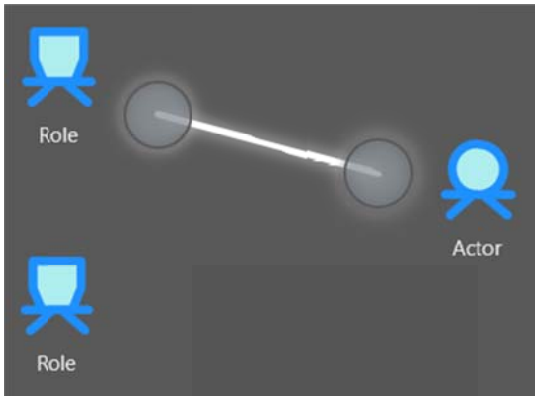


Figure 46: Connection selected and attached to elements.

Similar effects as the green glow of the Role in Figure 47 are applied to the groups and elements when elements are attached to groups, and when elements are removed from groups is a red glow that is applies to both elements and groups involved.

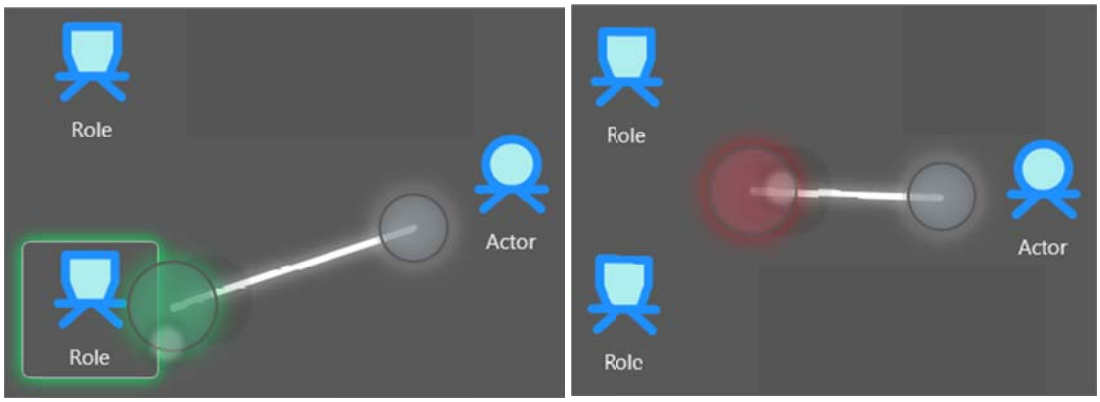


Figure 47: Connection attaching to an element (left) and detaching (right).

5.5 Challenges and limitations

As with most projects this one was not without its challenges. For instance the size of the screen which is a 12.1' was a challenge in itself because touch interaction requires for items to be bigger than in traditional GUI interfaces. As such the design of the buttons and elements had to be big enough for the use to hit and manipulate them easily.

As the software and the concepts such as Multi-touch, NUI and OCGM used in this project are very recent; this made finding resources and material difficult and sometimes impossible. Also touch and multi-touch area is a fast developing and expanding area, things change very fast and at times is hard to keep the pace.

Additionally the multi-touch and NUI concepts are somewhat complex which make them a bit hard to absorb. The touch and Surface APIs are also a bit complex if something other than the standard functionalities are required and consequently have a slightly long learning curve. These two create a challenge for someone who is starting a new project and has no previous experience with the APIs.

The prototype had to separate the elements that compose the diagram into several layers as described in section 5.3.3 Components. This imposed several challenges and limitations on the prototype. One of the challenges was that the connection had to be split

into two layers; the line that represented the relationship was in the connections canvas and the connections handles were in another. The two parts had to be kept synchronized, as one moved the other had to follow and vice-versa. This also happens with the selection and remove functionalities, as the elements are in one layer but the gestures and pen interaction functions on another. Then all the elements have to be represented in the layer where the gestures are recognized.

There were also some hardware limitations as the digitizer only recognizes two touch-points (in this case two fingers) at the same time. And there was also the limitation of the finger and the pen being mutual exclusive, as the touch was ignored when the pen was in use. This created some limitations at the level of possible affordances and interactions.

6 Validation - User Tests and Evaluation

During the development processes of the prototype tests were conducted to both guide and validate the interactions. The guiding tests consisted of several alternative interactions that where tested one versus the other to see which ones were more adequate to keep implementing. Those tests were performed in the first two stages of the user testing.

The third stage was dedicated to validate the prototype itself, by testing it against other forms of creating HAM sketches, namely the plain simple paper and the CASE tools MetaSketch. However, only were possible limited interface usability evaluations.

In this chapter only a brief description of the tests performed, the conclusions and design decision that they produced are going to be presented. The full user studies are on Appendix 3, where the tests performed, the task sheets, the hypothesis and the results are fully presented and explained.

6.1 First stage user tests

In this evaluation three functionalities were studied and for each there were two alternative interactions. Here are for each test a brief description of the alternatives, the conclusions from the test and the design decisions that those conclusions led to. The full description of the tests and the rest of the material are on the Appendix 3 section 1.

6.1.1 Test 1

This test had the objective of evaluating the addition of elements to the diagram. Therefore the user has to add elements from the toolbox and also draw those same elements using the pen. So here the two alternatives are then, adding elements using the finger with the toolbox and adding elements by drawing them with the pen. The hypothesis is that using the toolbox is faster and less prone to mistakes, as stated in hypothesis 1.1.1.2 in the Appendix 3.

6.1.1.1 *Conclusions*

Results show that for adding an element by dragging it from the pallet users took in average 41% the time that they took to draw an element; and taking into account the whole task of adding the three elements, when adding from the pellet users took only 66% the time to draw them.

More mistakes where expected, so the results from the mistakes metrics are a bit surprising as they showed almost no mistakes. This could be because the users where being careful on their drawings and because some users dragged the elements from the pallet using the pen instead of the fingers. Even so the count of mistakes for drawing is still higher than dragging.

This indicates that the hypothesis is confirmed and dragging elements from the toolbox seems to be the better of the two options.

6.1.1.2 *Design decision*

As a result of this test, the creation of new complex elements such as the actor or role will not be supported by drawing the elements themselves but using a pallet or panel from where the elements can be added.

6.1.2 Tests 2 and 3

Test 3 is a repetition of test 2; the difference is that in test 2 the elements are locked and in test 3 they are not locked. The user has to drag four elements which are in the middle of other elements all the way to the four borders of the diagram. This is to see if during panning the users will accidently displace another element. And hypothesis 1.1.2.2 from the Appendix 3 is that users will make more mistakes when the elements are unlocked than when they are locked.

6.1.2.1 *Conclusions*

The results obtained contradict the hypothesis, in the sense that users did more mistakes when the elements where locked than when they were unlocked, actually they did not do any mistake with elements unlocked. Users did not expect the elements to be locked, that could be because of the behavior of elements on other diagraming tools like Visio, in which elements are not locked.

The fact that there were no mistakes in test 3 alone does not indicate that the hypothesis fails. What leads to that conclusion is that users in test 2 still looked for empty space to preform panning actions, even after they knew that elements had to be unlocked to be moved.

The results obtained then point that having elements unlocked will not lead the user to making the mistakes referenced in the hypothesis, thus invalidating it.

6.1.2.2 *Design decision*

As a result of this test, elements will not be kept in a locked state. They will be unlocked so users do not have the need to unlock them to be able to manipulate them.

6.1.3 Tests 4 and 5

Test 5 is a repetition of test 4; the only difference is that in test 4 the pen's button operates as a press & hold and in test 5 the user only has to press the button once to enter selection or ink mode. These tests were designed so the user had to switch between selecting with the pen and other operations with the pen to maximize the use of the button. The hypothesis 1.1.3.2 of Appendix 3 is that using the button as a toggle is less effective and more propitious to mistakes than using the button as a press and hold.

6.1.3.1 *Conclusions*

Results show that users committed about four times more mistakes in test 5 than in test 4, thus confirming the tests hypothesis and indicating that press & hold should be used instead.

During the execution of the tests users had difficulty finding the button on the pen; sometimes they had to even look at the pen to see where it was. This might be because the button is not salient enough and so it is not easily graspable.

That allied with users saying that the cursors are not feedback enough for the current mode, suggests that there is need for another way of perceiving which mode the pen is and to switch modes as well. For that purpose having controls in the top toolbar that provide feedback and allow changing modes should be sufficient to solve the problem.

6.1.3.2 *Design decision*

As a result of this test, to select multiple elements with the pen users have to use the button as a press & hold and not as a toggle or a mode switcher. In addition to this there will be a control on the toolbar to give feedback to the users on which function they are currently on (ink, selection, and eraser). These controls besides giving this feedback can also control the function of the pen.

6.2 Second stage user tests

In this evaluation five functionalities were studied with the purpose of guiding the design and interaction validation. There were two alternative interactions for guiding the design and for validation only one interaction per functionality was tested. For the validation tests usability defects were gathered and not performance metrics as with the guiding tests.

6.2.1 Test 1 and Test 2

These tests have the objective of evaluating hypothesis 2.1.1.2 from the Appendix 3, which is that double-tapping is more efficient and will be prone to less user mistakes than scratching. So they focus on changing the names of the components. In Test 1 the user has to change the names starting by double-tapping the components. In Test 2 the user starts changing the names by scratching the current name of components.

The tests also have the objective of validating the Ink Recognition panel for writing the name of element. Afterwards the user has to use the ink recognition panel to write the new name using the pen and insert it into the component.

6.2.1.1 *Conclusions*

Results show that in test 2 users made triple the mistakes they made in test 1, this confirms the hypothesis and as expected double tapping an element is more efficient and is susceptible to less mistakes than scratching an element.

Now these results may be influenced by the facts that the scratch stroke has to be horizontal for it to be recognized and the users had difficulty understanding the gesture as they initially thought that it was a simple dash as in highlighting. Also all users had little to no past experience with tablet-pcs, however with the mouse is a different scenario as they all use it every day.

Regarding scratching the name there still is the fact that in the future with the implementation of several types of connections, and since some of them might have no name when created but the user could add a name subsequently, the implemented gesture should not be in function of the current name.

During the tests was observed that one user tried to write the name of the elements on top of the text box of the element, this could be because the textbox sends out the wrong affordance to the user of editing the text directly on the textbox, as it is a standard behavior. Also one user tried to start edition of another element immediately after writing down the name of one element. Putting these too observations together it seems that the user had perceived the affordance of being able to manipulate the elements on the diagram when writing the name.

Regarding defect 3, one user expected that tapping would insert the new name; however tapping in a clear space is the default escape action, and changing the tap to insert the new name would create inconsistency.

6.2.1.2 *Design decision*

From the results obtained and the observations made some design decisions surfaced:

- Scratching the name will be deprecated and double-tapping will be maintained
- Regarding defect 3 and 6 the background of the ink recognition panel will be more opaque so it does not send the affordance of manipulating the elements behind it
- Regarding defect 1 the "Recognize" button will be removed from the ink recognition panel
- Regarding defect 3 Tapping on the ink recognition panel will maintained as canceling ink edition
- Regarding defect 4 the line will be resized to be a bit bigger and its edges will gradually get transparent so it blends with the rest of the panel, also another line will be added to give the impression of being able to write in the hole panel
- A Pen/Ink icon will be added near the ruler lines to convey the affordance of Ink input to the user

The list of defects is on Appendix 3 section 2.2.2.2 Results.

6.2.2 Tests 3 and 4

Tests 3 and 4 relate to the hypothesis 2.1.2.2 form the Appendix 3, highlighting an element by underlining it should be less efficient than contouring it. So their only focus is on highlighting and lowlighting components. In test 3 the user has to highlight by contouring the components with the pen. On test 4 he has to underline the name of the elements to highlight the elements.

6.2.2.1 *Conclusions*

The results collected in these tests show that users committed no mistakes in contouring the elements and only two when underling the element, thus confirming the hypothesis. However the difference is not that big and once users picked up the underling gesture they said it made sense. But then again if an element has no name it cannot be highlighted using the underline.

The mistakes committed while underlining might be related to how the gesture is implemented, which is using the gestures Left and Right from the Windows InkCanvas. The problem is that they do not allow for much noise on the stroke which puts constraints on their use.

One user after making the tests tried highlighting multiple elements by contouring them and ended up grouping them, that was a mistake he made only once and learned grouping from it and started experimenting creating groups with other elements and highlighting other elements. Since creating groups was not a functionality tested at this stage, the user did not know that it was possible to create groups until then. This means that even if the two gestures are in conflict, they are sufficiently distinct that users can understand the difference and start using the gestures immediately. As some of the interactions are made through gestures which

have to be discoverable by the users, the fact that the user was experimenting with the interface is a benefit because means the interface is exploarble.

Regarding the connections, instead of users underlining the name they tried to overload the connection itself and some said that contouring a connection did not make much sense. Maybe it is because the result of contouring an element is adding an actual contour to the visual of the element and in a connection it is a different outcome since no contour is added, the effects are only applied to the current line. This might indicate that a different gesture should be used to highlight connections than the one used to highlight elements, maybe something more related to the end result of the highlight of the connection.

These are somewhat confusing results but they point to some aspects that have not been forethought, for instance the different approaches users make when highlighting connections vs. elements. And during the initial user studies when asked what they would do to highlight an element, users responded that they would contour or underline the element, so both gestures should be kept.

6.2.2.2 *Design decision*

Both the contouring of the element and underlining will be maintained as gestures to highlight elements.

However improvements on the underlining gesture will be made to provide the user with some more freedom and forgiveness when using this interaction.

A new gesture will be devised and implemented to highlight connections.

6.2.3 Tests 5 and 6

Tests 5 and 6 put to trial the hypothesis 2.1.3.2 form Appendix 3, that states that users would more accurately select using the pen, but on the other hand using the pen should take more time and commit more mistakes because of the button. As such the tests concentrate on selecting elements. On test 5 users have to select the elements using the pen to lasso around them and tap on an element to add it to the selection. Whether in test 6 the user has to use the finger to perform the selections.

6.2.3.1 *Conclusions*

Results are very similar on both tests, but as expected one user had a bit more difficulty on test 5 lassoing with the pen. However the lasso selection with the pen is a legacy interaction and it saves time when the user is interacting using the pen, as he has not to switch to the fingers to do the selection. So lasso selection with the pen should be maintained.

Tapping with the pen was an unexpected behavior by the users. It did not seem natural in the context of the diagram, since they could already tap using the fingers. However users did not commit any mistakes wen performing the taps and for the same reasons as with pen lasso selection it should be kept, those are because tapping is a legacy interaction used in tablets and if the user is interacting using the pen he does not have to switch to the finger to add an element to the selection.

About the fingers interactions, users did not commit any mistakes in the interaction. However they expected the controls on the toolbar not to function as full toggle buttons, in the sense that when they finished a selection they expected the control to be deactivated. It that would be the case, then the interface would be introducing hidden behavior and inconsistency as it would not be consistent with the other controls for the ink and rubber.

With the lasso selection either in the case of fingers or pen, it was observed during the test that users expected the elements selected to be added to an existing selection and not create a new selection. Now this is a surprising user expectation, since the standard behavior is that a lasso tool creates new selections. This aspect would need more study to make a more informed decision.

6.2.3.2 *Design decision*

As they are both useful depending on the context of interaction, both interactions will be kept implemented.

6.2.4 Test 7

The test has the objective of validating functionalities 2.1.4 Copy elements and 2.1.5 Remove elements from Appendix 3. To that purpose the user has to copy and paste the selection made on test 6. Then the user has to erase the group, all elements within and all the connections related to them from the original components.

6.2.4.1 *Conclusions*

When removing elements using the pen, there was no usability defects found. Furthermore it was observed that the performance of the interaction was better than expected, even with the option of removing elements and connections that are attached to a group not yet implemented. The performance was good enough to even consider not having that option implemented as conceptualized.

With copy of elements two usability defects were found, the defect 1 is about the visual effects of holding an element which is they get elevated. The defect is that only the element under the finger has the effect, where the full selection should have the effect. Defect 2 is about the elements that are pasted not being all selected as they should. This is a more serious defect because if the user wants to adjust the position of the elements, he has to select them again thus not being much efficient. The first defect is not as important as the selected elements all get copied despite not all having the held effect.

6.2.4.2 *Design decision*

The usability defects will be addressed, giving priority to defect 2. The activation effect will be added to all selected elements when one element of the selection is activated or held. And all elements that are pasted will become selected.

6.3 Third stage user tests

This stage is the final stage and was comprised of one major test which was the creation of a diagram in three different environments, but unfortunately this stage is limited as it was only possible the execution of one test with one user. After creating the diagrams the user answered a small questionnaire comprised of questions about the user's existing experience

with diagram creation and questions about the creation of the diagrams in the tools. The diagram, the questionnaire and the results from the test is in Appendix 3 section 3.

As the same diagram was asked to be created three times there could be issues of the user learning the diagram and become faster with each test. For it not to be an issue that impacts heavily on the outcome of the test, the user was asked to start the diagram from a different part for each of the three tools. Moreover the tools where so much different from each other that the user could not be affected by doing the same actions and eventually learning them and getting faster at them.

The results obtained in this stage are very limited and cannot be used to take definitive conclusions, however they yield some aspects that are useful for further tests and that give an insight of what to expect form them.

6.3.1 Conclusions

Results show that the prototype took more time than the paper and MetaSketch tests and also that the user committed more mistakes. However several factors contributed for this to happen, namely the performance of the prototype seemed to deteriorate considerably when using the screencast software which affected the touch responsiveness and the ink recognition timings. This performance drop only worsened when more elements were added and the diagram grew in size, however the same was not noticed with the MetaSketch. Those factors both contributed to a slower competition time and for the user to commit more mistakes as things were moving and responding slower that he expected.

Additionally the user said that he did not like to write on the Tablet-PC and was not too fond of the pen either, especially because of the barrel button that was unwillingly pressed.

Then after the user responded to the questionnaire and said he does not have much experience with human activity modeling, he commented that the elements were somehow confusing because they seemed all alike. It is true that there are some similarities within the elements from the participation map, as most have some sort of legs, arms and a head; and this can be confusing to someone who was not trained or have experience with the language. Maybe the differences between the widgets could be emphasized somehow, using colors or making the arms and legs longer.

About the questions about the user's evaluation of his experience with the tool, the user responded with values of 3 and 4, which means that in the user's opinion there are still some aspects that could be improved, but overall is good. When questioned of those aspects the user responded that they had to do with the pen itself and the button. That leads to the conclusion that this button, despite being useful gets in the way of the users and maybe should be deactivated. However to make any design decisions some more tests should be conducted.

7 Conclusion

Developers and designers make use of paper sketches to express their concepts, because of its almost constraint-free environment where they have as much freedom as they need. Though paper has some disadvantages which computer systems that can offer the same freedom people have from paper but none of the disadvantages and limitations can be overcome.

Developments on touch-sensitive screens with dual input digitizers now open the possibility of interacting with portable devices such as the tablet-pc using a pen as well as fingers. This provides the opportunity for more intuitive sketch applications that use a more effective, efficient and more natural combination of pen and fingers input than just finger or just pen interactions.

In this dissertation, as there was no specific application domain, a search was conducted to find one that was fitting and that provided innovation opportunities. From that search CASE tools was the elected domain, as it had sufficient work done that could be used as a starting point but yet still had space for improvement. In the CASE tools situation, it was the sketching interface that could be improved by adding the ability to manipulate the diagram components and the diagram itself using the fingers.

In this research some sketching and CASE tools where analyzed that provided useful information of the needs and requirements of a tool that used sketch input. However more information was needed to fully understand all the aspects of the problem and to start conceptualizing the system. Then user studies were conducted using a Model Driven Inquiry approach, preliminary models of the user activities where created that described the activities developers and designers practice when conceptualizing a system using paper and then CASE tools. Over those models scenarios, tests and questionnaires were devised and conducted which provided more information that could be added to the models completing them and providing more concrete knowledge.

With all the information gathered the system could then start to take shape as the concepts for the interactions started to be devised and developed into the prototype. These concepts were put to the test in the evaluations that revealed which concepts should be removed and which concepts and functionalities should be further developed. This was an iterative process as after the evaluations the concepts were revised or new concepts where created and introduced into the prototype for evaluation. This process had three iterations being that the last was more concerned with the validation of the interface and the interactions and not as much with guiding the development as the first two.

The development of the prototype occurred over the recent Microsoft touch technology, which is composed by the Windows 7 Touch API and the Surface Toolkit for Windows. Combining these two is advantageous as it provides the basic touch functionalities form Window 7 and the natural interactions and controls from the Surface Toolkit.

However that combination is not a silver bullet, there were still limitations and constraints that had to be dealt with. As this was a recent technology that indeed supports

very well touch interaction, finding resources to build upon is not always easy as with other well established technologies. Moreover this is to a certain extent an area in development and rapid expansion, and also the NUI paradigm is an extremely recent concept and the information available is sometimes hard to find or not available.

Hardware also imposed some limitations as the maximum number of touch-points was of two and the touch and the pen were mutually exclusive. This constrained the possible interactions and the affordances, which resulted in them having to be devised around those limitations.

With the results from the first two stages of the evaluations it was possible to refine the interactions and improve the interface itself. They also validated a few affordances and interactions, namely copying and removing elements. Because of those contributions these evaluations were an essential tool and played a crucial part in the development process, as they pointed and guided the process in the right direction.

Although only limited final evaluations were possible, some conclusions can already be drawn. The overall time for the completion of the diagram in the prototype took more time than the paper and MetaSketch tests, but specific factors contributed to that such as the use of the screen capture software that seemed to deteriorate considerably performance of the prototype. This affected the touch responsiveness and the ink recognition timings; additionally users said that they did not like to write on the Tablet-PC and that they had not much experience with human activity modeling so the elements somehow confused them because they seemed all alike.

The execution of more tests would bring more concrete data and influencing factors, which would allow for more solid conclusions.

7.1 Future work

As with all software projects and most things, there are aspects that can be improved and further developed, new functionalities can be added that better support developers express and model their ideas and concepts.

Further studying and evaluations of some of the functionalities could improve the overall usability and performance of the tool. For instance studying the lasso selection would allow finding from users which is the expected behavior, to add to a selection or to create a new selection. Also from the user studies it was found that the name insertion was somewhat hurting user performance, by studying it further it could bring a new perspective that would improve that.

The Ink Recognizer is not a “real component”, because it is embedded in the main window. However just by separating it a component could be created that can be reused in any other interface which when required collects the strokes and recognizes the handwritten text. Rethinking the ink recognizer into a standalone component could improve its performance and make the interface more efficient. Moreover the interaction with the handwriting input panel could also be refined to give the user a more smooth and effective

transition between elements manipulation and name edition, despite the current solution being less intrusive than the native ink input panel in Windows 7 which popped out when the cursor was placed in a text editable area.

By the conclusion of this document only limited usability evaluations were conducted, however they yielded valuable results and information. More of these evaluations could be conducted that would result in more and better results and directly impact the prototype.

Implementing MVC architecture would make the tool more robust and less change resilient. It would also separate the interface design and components from the rest of the prototype; this would make it easier to redesign the interface and would improve the performance of components such as a gesture or an ink recognizer which in turn would allow for a faster response and feedback also improving the performance of the interface.

Integration of a model generation component, most likely MOF, would be beneficial as it can represent the diagrams created in a formal model that could then be exported to other tools and further refined and reutilized. Giving support to the generation of XMI from the models would probably be the best option for exporting the model.

Once model generation is supported, widening the scope of support for more diagrams, languages and models would open opportunities for the tool to come into contact with new concepts and extend the interface to support them. Especially doing that for the remainder of HAM domain would be a valuable tool that would support full HAM modeling as it becomes more widespread and utilized by developers.

References

Aliakseyeu, D., Irani, P., Lucero, A., & Subramanian, S. (2008). Multi-flick: an evaluation of flick-based scrolling techniques for pen interfaces. *CHI 08 Proceeding of the twentysixth annual SIGCHI conference on Human factors in computing systems* (pp. 1689-1698). ACM.

Belis, M. (n.d.). *"Who Invented Touch Screen Technology?" in Inventors at About.com*. Retrieved August 2011, from http://inventors.about.com/od/tstartinventions/a/Touch-Screen.htm

Bett, B. (2007). *"History of CASE" at Toolbox.com*. Retrieved August 2011, from http://it.toolbox.com/wiki/index.php/History_of_CASE

Blake, J. (2010). *"Blake.NUI" at CodePlex*. Retrieved August 2011, from http://blakenui.codeplex.com/

Blake, J. (2011). *Natual User Interfaces in .NET* (Manning Early Access Program ed.). Greenwich, Connecticut, United States of America: Manning Publications Co.

Buxton, W. (1986). Chunking and Phrasing and the Design of Human-Computer Dialogues. *Proceedings of the IFIP World Computer Congress*, (pp. 475-480). Dublin, Ireland.

Buxton, W. (2007). *"Multi-Touch Systems that I Have Known and Loved" at billbuxton.com*. Retrieved August 2011, from http://www.billbuxton.com/multitouchOverview.html

Buxton, W. (2010). *"CES 2010: NUI with Bill Buxton" at MSDN's Channel 9*. Retrieved August 2011, from http://channel9.msdn.com/posts/LarryLarsen/CES-2010-NUI-with-Bill-Buxton/

Chen, Q., Grundy, J., & Hosking, J. (2003). An E-whiteboard Application to Support Early Design-Stage Sketching of UML. *Proceedings of 2003 IEEE Human-Centric Computing Conference*, (pp. 219-226). Auckland, New Zealand.

Chung, R., Mirica, P., & Plimmer, B. (2005). InkKit: a generic design tool for the tablet PC. *CHINZ "05 Proceedings of the 6th ACM SIGCHI New Zealand chapter"s international conference on Computer-human interaction: making CHI natural* (pp. 29-30). ACM.

CollabNet, Inc. (n.d.). *"Welcome to ArgoUML" in Projects at Tigris*. Retrieved August 2011, from http://argouml.tigris.org/

Constantine, L. L. (2009). Human Activity Modeling: Toward A Pragmatic Integration of Activity Theory and Usage-Centered Design. In A. Seffah, J. Vanderdonckt, M. C. Desmarais, A. Seffah, J. Vanderdonckt, & M. C. Desmarais (Eds.), *Human-Centered Software Engineering* (Vol. VIII, pp. 27-51). London, United Kingdom: Springer London.

Darnton, G. (2001). *PSL/PSA - History.* Retrieved August 2011, from "Problem Statement Language/Problem Statement Analyzer" at pslpsa.com: http://www.pslpsa.com/HistoryTop.htm

Darses, F., Mayeur, A., Elsen, C., & Leclercq, P. (2008). Is there anything to expect from 3D views in sketching support tools? DCC'08. *Proceedings of the 3rd International Conference on Design Computing and Cognition.* Atlanta, USA.

Dickinson, J., Yu, Z., Zeng, Y., & Antunes, H. (2005). Pen-tablet as a CAD interface alternative. *Robotics and Computer-Integrated Manufacturing, 21*, pp. 465-474.

Everet, A. (2010). *"Types of Tablet PCs" at EzineArticles*. Retrieved August 2011, from http://ezinearticles.com/?Types-of-Tablet-PCs&id=4100635

Feldkamp, O. (2009). Hands-on Lab "Windows 7 Multitouch". Switzerland.

George, R. (2009). *"Terminology: the difference between a gesture and a manipulation" at Ron George Blog.* Retrieved August 2011, from http://blog.rongeorge.com/design/interaction-design/terminology-the-difference-between-a-gesture-and-a-manipulation/

George, R. (2009). *"Welcome to the OCGM Generation! Part 2" at Ron George Blog*. Retrieved August 2011, from http://blog.rongeorge.com/design/welcome-to-the-ocgm-generation-part-2/

Hammond, T., & Davis, R. (2002). Tahuti: A Geometrical Sketch Recognition System for UML Class Diagrams. *2002 AAAI Spring Symposium on Sketch Understanding.*

Hammond, T., Gajos, K., Davis, R., & Shrobe, H. (2002). Sketch Recognition in Sfotware Design.

Hewlett-Packard Development Company, L.P. (2007). *"HP Digitizer Pen for HP TouchSmart tm2 Series Tablet PC" at HP - Home & Home Office.* Retrieved August 2011, from http://www.shopping.hp.com/product/computer/categories/specialty_products/1/accessories/VX406AA

Hewlett-Packard Development Company, L.P. (n.d.). *"PC Notebook HP TouchSmart tm2-2150ep (XD811EA) - especificações e garantia" at HP*. Retrieved August 2011, from http://h10010.www1.hp.com/wwpc/pt/pt/ho/WF06b/321957-321957-3329744-3835813-3835813-4247583-4315585.html

Holwerda, T. (2009). *"Tablets: Sought by Nobody, Hyped by Everybody" at OSnews*. Retrieved August 2011, from http://www.osnews.com/story/22287/Tablets_Sought_by_Nobody_Hyped_by_Everybody

Holwerda, T. (2010). *"A Short History of the Tablet Computer" at OSnews*. Retrieved August 2011, from http://www.osnews.com/story/22739/A_Short_History_of_the_Tablet_Computer

Igarashi, T., Matsuoka, S., & Tanaka, H. (1999). Teddy: a sketching interface for 3D freeform design. *ACM SIGGRAPH 2007 courses* (pp. 409-416). ACM.

Juchmes, R., Leclercq, P., & Azar, S. (2006). A Multi-Agent System for the Interpretation of Architectural Sketches. *Special issue of Computers and Graphics Journal, 29*.

Kay, A. C. (1972). A Personal Computer for Children of All Ages. *Procceding of the ACM National Conference.* Boston.

Kieffer, S., Coyette, A., & Vanderdonckt, J. (2010). User interface design by sketching. *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems EICS 10* (pp. 57-66). ACM Press.

Landay, J. A., & Myers, B. A. (2001). Sketching interfaces: toward more human interface design. *Computer, 34*, 56-64.

Ley, M. (2011). *"DBLP: Daniel Teichroew" at Universität Trier*. Obtido em August de 2011, de http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/t/Teichroew:Daniel.html

Liwicki, M., Rostanin, O., El-Neklawy, S. M., & Dengel, A. (2010). Touch & Write — A Multi-Touch Table with Pen-Input. *DAS '10 Proceedings of the 9th IAPR International Workshop on Document Analysis Systems* (pp. 479-483). New York, New York, USA: ACM New York, NY, USA ©2010.

Malysa, S. (2010). *"Types of Tablet PCs" at eHow*. Retrieved August 2011, from http://www.ehow.com/list_7436158_types-tablet-pcs.html

Matos, J. M. (2008). MetaSketch OCL Interpreter. *Dissertação para obtenção do Grau de Mestre em Engenharia Informática*. Funchal, Portugal: Universidade da Madeira.

Microsoft. (n.d.). *"Surface Toolkit for Windows Touch Beta Overview" at MSDN*. Retrieved August 2011, from http://msdn.microsoft.com/en-us/library/ee957352(v=surface.15).aspx

Moscovich, T. (2006). Multi-touch interaction. *Proceedings of the ACM Conference on Human Factors in Computing Systems* (pp. 1-4). ACM Press.

Nobrega, L., Nunes, N. J., & Coelho, H. (2006). The Meta Sketch Editor. *Proceedings of CADUI*, (pp. 201-214). Funchal.

Norman, D. A. (1988). *The psychology of everyday things.* USA: Basic Books.

Object Managment Group, Inc. (2011). *"OMG's MetaObject Facility" at OMG*. Retrieved August 2011, from http://www.omg.org/mof/

PAPERBOY, T. (2009). *"Courier: First Details of Microsoft's Secret Tablet" at Gizmodo*. Retrieved August 2011, from http://gizmodo.com/5365299/courier-first-details-of-microsofts-secret-tablet

Parenteau, L. (2009). *"UML modeling tools reviews : ArgoUML" at Software Development with Linux*. Retrieved August 2011, from http://laurentparenteau.com/blog/2009/03/uml-modeling-tools-reviews-argouml/

Plimmer, B., & Freeman, I. (2007). A toolkit approach to sketched diagram recognition. *Proceedings of the 21st British CHI Group Annual Conference on HCI 2007 People and Computers XXI HCI but not as we know it. 1*, pp. 205-213. British Computer Society.

Smith, G. M. (2004). The radial scroll tool: scrolling support for stylus- or touch-based document navigation. *Symposium on User Interface Software and Technology, 6*, 1-4. 2011, August.

Sun, L., & Guimbretière, F. (2005). Flipper: a new method of digital document navigation. *CHI05 extended abstracts on Human factors in computing systems*, 2001-2004. 2011, August: ACM.

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science, 12*, pp. 257-285.

Treadaway, C. (2009). Translating experience. *Interacting with Computers, 21*, 88-94.

van Lewen, J. (2010). PROPOSTA: A007 - Touch-n-Sketch Pen and fingers on a multi-touch sketch application for tablet PC's. Funchal.

Visio Team. (2009). *"Update: Visio 2007 Service Pack 2 and Visio Conference 2008" at Visio Insights*. Retrieved August 2011, from http://blogs.msdn.com/b/visio/archive/2009/05/11/follow-up-visio-2007-service-pack-2-and-visio-conference-2008.aspx

Visual Paradigm International. (2009). *"Freehand Shape Support" at Visual Paradigm*. Retrieved August 2011, from http://resources.visual-paradigm.com/index.php/visual-modeling/67-vm-general/354-freehand-shape.html

Visual Paradigm International. (n.d.). *"Class diagram" at Visual Paradigm*. Retrieved August 2011, from http://images.visual-paradigm.com/screenshots/vpuml80/windows/class_diagram.png

Visual Paradigm International. (n.d.). *"Solutions" at Visual Paradigm*. Retrieved August 2011, from http://www.visual-paradigm.com/solution/

Wacom. (2007). *"EMR® (Electro-Magnetic Resonance) Technology" at Wacom-Components.* Retrieved August 2011, from http://www.wacom-components.com/english/technology/emr.html

Wikipedia Foundation, Inc. (2002). *"Meta-Object Facility" at Wikipeida*. Retrieved August 2011, from http://en.wikipedia.org/wiki/Meta-Object_Facility

Wikipedia Foundation, Inc. (2004). *"Touchscreen" at Wikipedia.org*. Retrieved August 2011, from http://en.wikipedia.org/wiki/Touchscreen

Wikipedia Foundation, Inc. (2004a). *"Computer-adided software engineering" at Wikipedia.org*. Retrieved August 2011, from http://en.wikipedia.org/wiki/Computer-aided_software_engineering

Wikipedia Foundation, Inc. (2006). *"Tablet computer" at Wikipedia*. Retrieved August 2011, from http://en.wikipedia.org/wiki/Tablet_computer#cite_note-Gray-2

Wikipedia Foundation, Inc. (2007). *"Microsoft Surface" at Wikipedia*. Retrieved August 2011, from http://en.wikipedia.org/wiki/Microsoft_Surface#Surface

Wikipedia Foundation, Inc. (2009). *"iPad" at Wikipedia*. Retrieved August 2011, from http://en.wikipedia.org/wiki/Ipad

Appendices

Appendix 1 – Observations and Interviews

Appendix 2 – Concepts and Affordances

Appendix 3 – User Tests and Evaluations

Appendix 1: User Studies

1 Observations and Interviews

1.1 Plan

This list was based on Benyon and the Turners' evaluation guide (Benyon, Turner, & Turner, 2005).

	Step	Notes
1.	Based on a **scenario** devise a list of **tasks** for the users to perform. Make a list for sketching on paper and for the MetaSketch, if there is a necessity for two lists.	The tasks must be realistic, do-able with the paper and with the MetaSketch, and explorative of the systems.
2.	**Estimate** how long will it take for a user to complete the tasks.	Allow 50% more than the estimated execution time of the tasks per session (Benyon, Turner, & Turner, 2005).
3.	Prepare a **task sheet** for the users. Also prepare a set of **questions** for the interviews.	Be specific and explain so the user can understand them. Questions should be based on the tasks but at the same time exploratory (open-ended).
4.	Get **ready** for the observations and interviews.	Have the systems ready for the user to start his performance. Fresh paper for the sketching and the software running for the application observations. Have a note book and pens available for taking notes. Set audio and video recording equipment.
5.	Tell the user that this is an observation of the interaction and that it is **OK** and normal for him to make mistakes. **Explain** the procedure and **introduce** the **tasks**.	If it is a group of users will be harder to observe them all at the same time, so focus on the one doing the sketching or interacting with MetaSketch. At this point start the recording equipment.
6.	Users **start** the tasks. Tell users to give a **running commentary** of what they are doing, why and difficulties or uncertainties they encounter	Take notes of where users find problems or something unexpected and what it was, use time stamps if possible. Observe quietly, out of peripheral view and no signs. Do not interfere unless the user is stuck or have them move on to the next task, if possible. When possible check that the recording devices are working properly.
7.	When user is finished, **stop** the recordings and **save** the observation. Quickly **review** the notes and the questions.	Stopping the recordings and saving them gives less chance of something going wrong with the files. Reviewing the notes and questions, helps to find pertinent new questions about issues observed.
8.	Start a **new** recording for the	In the beginning stick to the questions to get the

interview audio or video. Have the questions at **hand** and start the interview.	interview going, when convenient explore the answers of the user or explore new topics. For a group of users a simple questionnaire might be helpful.
9. Quickly **review** the answers to tie any lose ends. **Thank** the users for their help.	
10. Write up notes **quickly.**	So you do not forget any important connections or details. Incorporate them as soon as possible into a study report.
11. **Stop** the recording equipment and **save** the files.	
12. Restart form the 4th step for a new user/group	

1.2 Scenario

The scenario used was about a normal operation in an ATM and it was devised because it is a widely used example in courses about data manipulation and software engineering, and as the test subjects where software engineering students they would already be familiar with the scenario and could focus on doing the activity models.

> *« John needs money to go to lunch. So he goes to an ATM to withdraw some cash, inserts his card and enters the PIN when asked. He is not sure how much money is still left in his account, so John picks the option "Check balance" before making a withdrawal. He wants to check it on the monitor so he selects "Check on screen". John has more than enough for his lunch so he proceeds by choosing "Make another operation". The machine asks for the PIN again and John enters it. Then he chooses "Withdraw money" and then enters the quantity "10€". Finally he selects "Remove card". The card comes out and he gets it from the machine. Afterwards he gets the money and then the receipt. »*

This is the complete initial scenario, the scenario 0, the bigger one. This scenario tries to have the most common elements of the activity model in sufficient quantity for the tests, being those elements the activity and the task widgets and the relationships between them.

Anyhow from here as explained before, the task sheet was composed that included the smaller version of the scenario, the tasks for the user and the questionnaire. A second version of the sheet was created that contained an even smaller version of the scenario and required less tasks form the user and consequently less questions. They are both further down in the Task Sheets section.

1.3 Task list

This is the task list that was created from the scenario, and concerns the paper sketch:

1. Add the activities: "AC 1 - Withdraw cash for lunch" and "AC2 - Checking balance". The first uses the second, so also draw a uses connection between them.
2. Add the task "T1 - Check balance". It is related to AC2.
3. Add the tasks: "T2 - Identify myself", "T3 - Get my card", "T4 - Get money" and "T5 - Get receipt". They are all related so cluster them together, also they are related to both activity AC1 and AC2.
4. Add a border around those tasks and name it "common". Connect it to AC1 and to AC2.
5. Add the task "T6 - Withdraw money" near the activity AC1, because they are connected.
6. Add a connection: Task T6 uses the task T3.
7. There is precedence between tasks T3, T4 and T5. Add a “precedes” connection from T3 to T4 and another form T4 to T5.
8. Task T2 precedes tasks T1 and T6, add the connections. Also highlight task T2 to call to the attention.
9. Add a connection: Task T1 uses the task T3.
10. Ups, in the case of just checking the balance, one does not get any money. So T4 is not actually clustered with the others, it is with T6. Fix it.
11. And when just checking, one might have to get a receipt after getting the card, so there is a “precedes” connection form T3 to T5.
12. Add a border around T6 and T4, since they are now clustered. And name it "money withdrawal". Now activity AC1 is related to that cluster and not just with T6. Fix it.
13. Add a note to task T1 saying: Do not forget to put the options of "Checking on the receipt" and "Checking on the screen" on the task case.

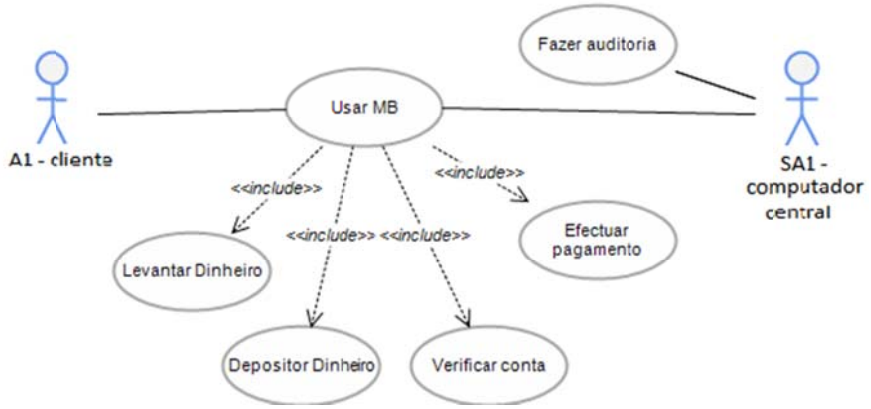
This list was devised with the purpose of covering the usual tasks people do when sketching, that is adding new components to the sketch and doing corrections. Also this covers some of the features that we wanted to observe users do, which are annotating, highlighting components and doing major corrections.

To do those corrections tasks 10 and 12 create problems for the test subject so he has to move components to another part of the sketch or even remove them. Obviously removing things form a paper sketch can be hard if using a pen, so we wanted to see if the subject would scratch out elements and add the new corrected elements on another part of the sketch, or if he would press harder on the pen or multi-stroke the corrections over and used the older component. Or maybe the subjects would do something different.

With the same kind of objective, after users did the tasks in the list, they were asked to pass a previously drawn paper sketch to a Use Case diagram, some using Meta Sketch and some Microsoft Visio.

The objective here was to see how users added, named and positioned the elements, if they first added all the elements and then named them, or if they named them as they added them, the same with the positioning.

The following image is a diagram created by one of the users using Meta Sketch and is a direct representation of the paper sketch that was presented to them during the tests.



1.4 Questionnaire

These are the questions that composed the questionnaire in the task sheets. They are presented here with a purpose and a set of expected answers from the users.

There is a different questionnaire for the sketching and diagramming parts of the tests.

1.4.1 About the sketching:

Q1 - What do you do first when you draw an activity/task? And then? Why in that order?	
Purpose:	To see if the user writes the name first or draws the widget.
Expected answer:	First I write the name and then I draw the circle/square around it. Because I cannot predict how big the name will be.
Q2 - What do you do first when you have to connect and draw two elements in a sketch? And then? Why do you do it like that?	
Purpose:	Trying to establish the order which the user draws the elements, if the he draws both elements first and then connects them, or if he draws one element and then a connection and then the other element.
Expected answer:	First I draw the elements and then the connection. Because I do not know the size of the elements and their positions.
Q3 - Why did you scratched/erased the task "T4 – Get money" and its connections?	
Purpose:	Find out why the user chose the particular action scratch or the particular action erase. To see why did not he choose the other option. And also to see if the user is not worried with the beautifulness of the diagram at this stage.
Expected answer:	Because it said on the tasks and … Using Pen → Scratched: … I was sketching with a pen, so I could not use a rubber and there is no corrector. Using Pencil → Erased: … I could, I was sketching with a pencil and rubber. Using Pencil → Scratched: … this is just a sketch it is not the final drawing, so it does not matter if it is not perfect or pretty.
Q4 - What if you were sketching in a multi-touch how would you remove an element?	
Purpose:	See if the user would do the same action as in the previews question or if he thinks of a new way for doing it.
Expected answer:	I would do the same.

Q5 - How do you normally represent a border? What do you do first when drawing it? … then?	
Purpose:	Find other notations, other than the traced border. And to see if the user first writes the name or draws the border.
Expected answer:	By drawing a line around the things (elements), like I did on the task. First I draw the line and then write the name.
Q6 - If you had to remove the border "common", what would you do? And then?	
Purpose:	To expose the problem of having to reconnect all the cut connections again or the problem of ending up with a big scratched rectangle in the middle of the diagram. Preparation for the next question, basically.
Expected answer:	Drawn with a Pencil: I would erase it with a rubber. And then … I would have to complete the connections again. Drawn with a Pen: Hum … Ups I cannot remove it … I would scratch it. It is just a sketch.
Q7 - What if you were using a sketch tool in a multi-touch screen, what would make it easier?	
Purpose:	Find ways of better supporting user action.
Expected answer:	Previews answer → Pencil: If the connections were not erased. Previews answer → Pen: If I could select the border and just delete it.
Q8 - In a paper sketch you just write the name of an element with a pen/pencil. What if you were using a sketch-tool on a multi-touch screen how do you think you would normally name an element?	
Purpose:	To find out if the user would still use the pen and just write it, or if he would use another method.
Expected answer:	I would use a virtual keyboard.
Q9 - How do you normally highlight an element?	
Purpose:	Find out which action is more commonly used to highlight an element. Making the edge of that element thicker or its name, shadowing, using colors …
Expected answer:	Normally I just circle it a lot until it is pretty thick.
Q10 - What if you were using a sketch-tool on a multi-touch screen would you do it differently? Why?	
Purpose:	Check if the user would transport his normal action to the sketching tool. That could be a gesture to use in the sketching tool.
Expected answer:	No. Because I am used to doing it this way.
Q11 - If you could sketch on a multi-touch screen would you prefer to draw the widget or drag it from a tool bar? Why?	
Purpose:	To see the user preference in relation to drawing or having something drawn by the computer.
Expected answer:	Drag it, because I would not like to draw the same thing over and over again.

1.4.2 About the widget-based diagram tool:

Q1 - What do you do first to add an activity/task? And then? Why in that order?	
Purpose:	Find out if the position is the first priority or the name.
Expected answer:	First I position it roughly, then I type the name and then I put it where I

	want it. Because like that I do not forget the name and it is easier to find the correct position afterwards, the size is set already and the name helps to relate to other elements.
Q2 - What do you do first when you have to add and connect two shapes? And then? Why?	
Purpose:	Trying to establish the order which the user add the elements, if the he adds both elements first and then connects them, or if he adds one element and then a connection and then the other element.
Expected answer:	First I add the shapes and then the connection. Because I need to have the elements to be able to connect them.
Q3 - How do you normally highlight an element of a diagram, what would you do first? And then?	
Purpose:	Find out which action is more commonly used to highlight an element. Making the edge of that element thicker or bolding its name, add a shadow, use colors for the background or the lines or name …
Expected answer:	Normally I just make the whole thing bold or fill it with some color.

1.5 Estimated times

These where times that where estimated for the steps form the plan and based on Scenario 0 presented above, the estimation was based on Benyon and the Turners' evaluation guide (Benyon, Turner, & Turner, 2005).

Step	Estimate Time (mm:ss)
4. Get **ready** for the observations and interviews.	01:00
5. Tell the user that this is an observation of the interaction and that it is **OK** and normal for him to make mistakes. **Explain** the procedure and **introduce** the **tasks**.	05:00
6. Users **start** the tasks. Tell users to give a **running commentary** of what they are doing, why and difficulties or uncertainties they encounter	10:18 + 50% = 15:27 ≈ 15:30
7. When user is finished, **stop** the recordings and **save** the observation. Quickly **review** the notes and the questions.	01:30
8. Start a **new** recording for the interview audio or video. Have the questions at **hand** and start the interview.	Sketch ≈ 15:00 Diagram ≈ 5:00
9. Quickly **review** the answers to cut any lose ends. **Thank** the users for their help.	02:00
10. Write up notes **quickly**.	01:00
11. **Stop** the recording equipment and **save** the files.	00:30
12. Restart form the 4th step for a new user/group	

Type	Total Estimated Time / User	Total for 3 users
Sketch	0:41:30	02:04:30
Diagram	0:31:30	01:34:30
Total:	01:13:00	03:39:00

Total was estimated for three users because it was indicated in (Benyon, Turner, & Turner, 2005) that testing or observing three for each role users would be enough to collect sufficient data.

2 Task Sheets

These are the task sheets that where given to the users during the user studies.

2.1 Task Sheet v1 – Scenario 1

You are a Software Engineer/HCI Designer, and you have to model an ATM (Automatic Teller Machine).

Scenario:

> *John needs money to go to lunch. So he goes to an ATM to withdraw some cash, inserts his card and enters the PIN when asked. He has more than enough for his lunch so he proceeds by choosing "Withdraw money" and then enters the quantity "10€". Finally he selects "Remove card". The card comes out and he gets it from the machine. Afterwards he gets the money and then the receipt.*

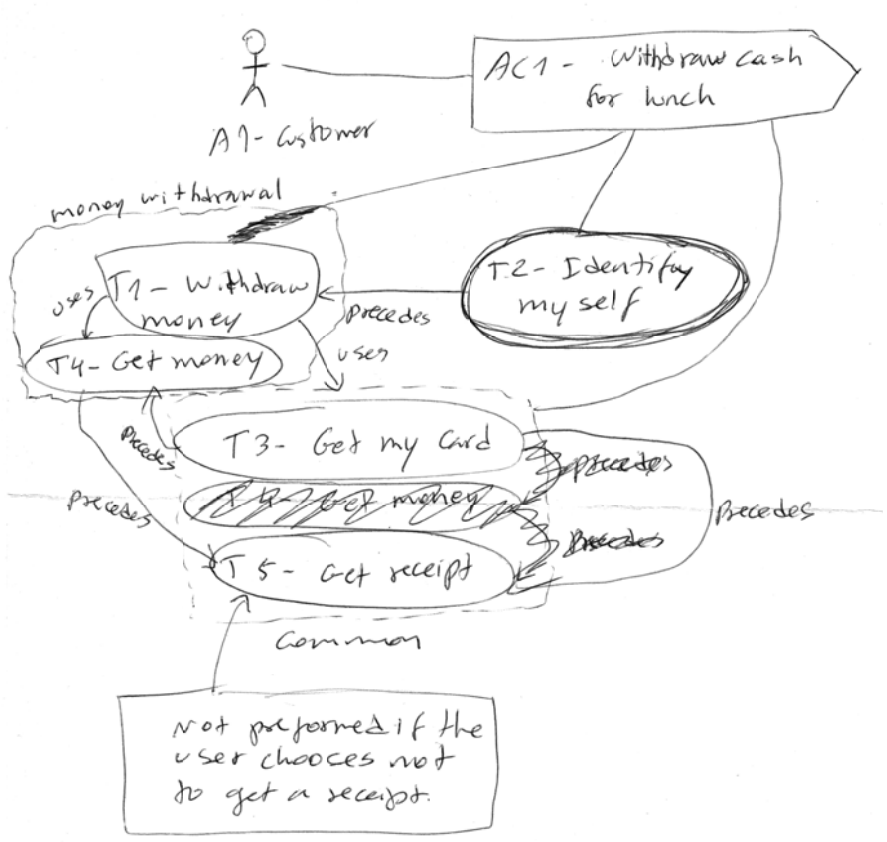
Do the model based on the scenario. Follow the task list bellow to complete the sketch:

Task List:

1. Add the actor "A1 - Customer" who is connected to the activity "AC1 - Withdraw cash for lunch" which you will have to add as well.
2. Add the tasks "T1 - Withdraw money" and "T2 - Identify myself" which are connected to "AC1 - Withdraw cash for lunch".
3. Task "T2 - Identify myself" precedes task "T1 - Withdraw money", add the connection. Also highlight task "T2 - Identify myself" to call to the attention.
4. Add the tasks: "T3 - Get my card", "T4 - Get money" and "T5 - Get receipt". They are all related so cluster them together.
5. Add a “precedes” connection from "T3 - Get my card" to "T4 - Get money" and another form "T4 - Get money" to "T5 - Get receipt".
6. Add a border around those tasks and name it "common". Connect it to "AC1 - Withdraw cash for lunch".
7. Add a connection: Task "T1 - Withdraw money" uses the cluster "common".
8. Oops, you can only get any money if it is a withdrawal. So "T4 - Get money" is not actually clustered with the others, it is with "T1 - Withdraw money". Fix it.
9. Do not forget that "T1 - Withdraw money" now uses "T4 - Get money".
10. One might just get a receipt after getting the card, so there is a “precedes” connection form "T3 - Get my card" to "T5 - Get receipt".
11. Add a border around "T1 - Withdraw money" and "T4 - Get money" and name it "money withdrawal".
12. Now activity "AC1 - Withdraw cash for lunch" is related to that cluster and not just with "T1 - Withdraw money". Fix it.
13. Add a note to task "T5 - Get receipt" saying: "Not preformed if the user chooses not to get a receipt".

Note: The questions asked to the users with this version of the task sheet were the questions that are in section 1.4 Questionnaire.

2.1.1 Expected sketch



2.2 Task Sheet v1 – Scenario 2

You are a Software Engineer/HCI Designer, and you have to model an ATM (Automatic Teller Machine).

Scenario:

John needs money to go to lunch. So he goes to an ATM to withdraw some cash, inserts his card and enters the PIN when asked. He has more than enough for his lunch so he proceeds by choosing "Withdraw money" and then enters the quantity "10€".

Do the model based on the scenario. Follow the task list bellow to complete the sketch:

Task List:

1. Add the actor "A1 - Customer" who is connected to the activity "AC1 - Withdraw cash for lunch" which you will have to add as well.
2. Add the tasks "T1 - Withdraw money" and "T2 - Identify myself" which are related so put them near each other.

3. Task "T2 - Identify myself" precedes task "T1 - Withdraw money", add the "precedes" connection.
4. Create a border around the two tasks and name it "money withdrawal".
5. Connect it to "AC1 - Withdraw cash for lunch".
6. Oops, task "T2 – Identify myself" may not be used in other situations other than the withdrawal of cash. So it is no longer part of the group "money withdrawal", put it outside of the group.
7. Connect task "T2 – Identify myself" to the activity "AC1 – Withdraw money for lunch".
8. Remove the border "money withdrawal".
9. Highlight task "T2 - Identify myself" to call to the attention.
10. Add a note to task "T2 – Identify myself" saying: "Also used in other situations other than money withdrawal".

Questions about the sketch

Q1 - What do you do first when you draw an activity/task? And then? Why in that order?

Q2 - What do you do first when you have to connect and draw two elements in a sketch? And then? Why do you do it like that?

Q3 - Why did you scratched/erased the task "T2 – Identify myself" and its connections?

Q4 - What if you were sketching in a multi-touch how would you remove an element?

Q5 - What do you do first when drawing it? … then? Why?

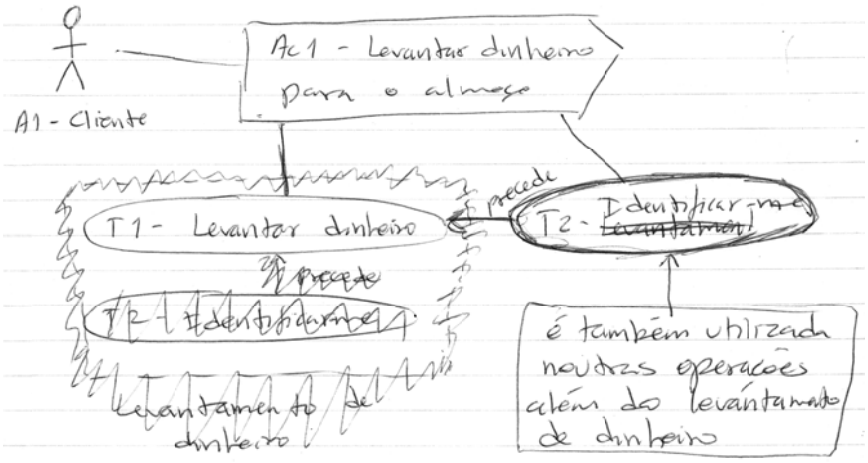
Q6 - What if you were using a sketch tool in a multi-touch screen, what would make remove the border "money withdrawal" easier?

Q7 - In a paper sketch you just write the name of an element with a pen/pencil. What if you were using a sketch-tool on a multi-touch screen how do you think you would normally name an element?

Q8 - What if you were using a sketch-tool on a multi-touch screen would you highlight an element in a different way that you used here in the paper? Why?

Q9 - If you could sketch on a multi-touch screen would you prefer to draw the widget or drag it from a tool bar? Why?

2.2.1 Expected sketch



3 Observation Notes

Regarding the diagram, users that participated in the tests with Scenario 1 created the Diagram using Microsoft Visio, and with Scenario 2 users used Meta Sketch.

3.1 Scenario 1

3.1.1 User 1

Sketch observations

- The first connection was forgotten
- First the user draws the contour of the activity and then he writes the name
- The name of the border was written inside of the border
- Scratched both tasks T1 and T4 and first created a border/group, only after created inside that border the tasks T1 and T4 again
- Added again a border to group T1 and T4
- The note was made in form of a balloon connected to T5
- After having scratched T1, the connection to "common" still persisted

Diagram Observations

- First added the number of need cases
- Then wrote the names of the cases and then grouped them

3.1.2 User 2

Observations regarding the sketch:

- Drew the contour of the activity AC1 before writing the name
- Drew the elements and then connected them
- Wrote the ID (ex: T1) of the elements inside them and the name outside, underneath them
- Left more space between the elements after realizing that there could be more connections
- First did the connection, then drew the arrow tip

Diagram Observations

- Added the number of need elements then gave their names

3.1.3 User 3

Observations regarding the sketch:

- First wrote the names and then drew the elements
- When adding several elements, first writes the name first leaving some space between them and then draws the elements around the name
- To move task T4 first draws the element in the new place and then scratches the old one
- Writes the note text and then draws the contour
- When wrote the wrong name, scratches it and writes the correct one

3.2 Scenario 2

3.2.1 User 1

Observations regarding the sketch:

- First drew an actor and then made a connection. Afterwards wrote the name of the activity and then drew the contour
- When drawing multiple tasks first the names and then draws the contours
- Connected the tasks to the actor, even if it was not asked
- First created the border, then gave it its name
- When removing the task T2, scratched the border around T2 and closed it, just leaving T1 inside
- The connection T2 with AC1 was highlighted because it ran over sketch elements that were scratched
- When adding the note first draws the connection and then writes the note, but does not add a contour to it

Diagram Observations:

- Added several Use Cases then gives them names, and puts them more or less in position
- The connections only appear if the elements are moved around and positioned further apart

3.2.2 User 2

Observations regarding the sketch:

- Gave the subject's name from the scenario, to the actor in the sketch
- First drew the elements and then made the connections
- First drew the contour of the activity and then gave its name
- In the tasks was the other way around
- To solve the task T2 problem decided to erase the tasks and the border, and draw them again in another area
- When adding the note drew part of the contour and then wrote the note; finally finished the contour

Diagram Observations:

- First added the elements, starting from the top of the sketch
- Positions them in the desired position and them gives them names
- In the end makes all the connections

4 Conclusions from the Observations and Interviews

This is a set of conclusions on different aspects that aroused form the observations and interviews conducted during the tests:

- When a contour is drawn and then the name is added, if the name is too big to fit inside the shape, then the shape should grow.
- The "sheet of paper" should be infinite in the sense that when adding a new element, if there is no more space near the adding area, one should just be able to drag the diagram a bit more to get some clear space.
- Support the removal of elements by gestures or the traditional menu option, users are somewhat divided. Maybe support both features.

- When removing an element leave the existing connections or remove them, some say keep others remove, maybe provide a choice to the user. This also happens for the borders and their elements.
- By dragging an element and dropping it near the end of an open connection, it should connect automatically.
- Majority of the users used a thicker line for highlighting an element, but some still consider adding color to the background.
- For more complex shapes like the actor, users tend to pick from a tool bar or from some pane than draw them by hand.
- To name the elements users tend to pick the virtual keyboard and not the pen even doe they do not discard it.
- When adding multiple elements, users first write their names and then draw the contours or in the diagraming tool they first position all the elements and then write their names. Only then they connect them.
- The system should recognize that elements belong to a border when one is drawn around them.
- Users did not found important to scratch an element to call the attention that it was not supposed to be there any more, they just scratched them because they needed to get rid of them.

References

Benyon, D., Turner, P., & Turner, S. (2005). *Designing Interactive Systems: People, Activities, Contexts, Technologies.* Addison-Wesley.

Appendix 2: Concepts and Affordances

1 Paper Concepts

These are the initial concepts done to start imagining what would be the interaction with the prototype. As happens with other areas, the better way to start doing those concepts was though paper sketches. As previously mentioned they allow us to have as much freedom as possible, not constraining creativity and are a quick way to go through several concepts.

The concepts are gathered into groups of tasks, as the several concepts are presented within them. For instance in Connections are presented the interactions for creating, attaching/detaching and rerouting connections.

1.1 Navigation

Panning which is moving around in the diagram sketch can be done with the usual drag or with a flick gesture as in Figure 1.

Figure 1: Concept - Navigational Flick Concept.

Zoom In and Out could be done by moving two touch devices towards or away from each other, that is the well-known pinching as in Figure 2.

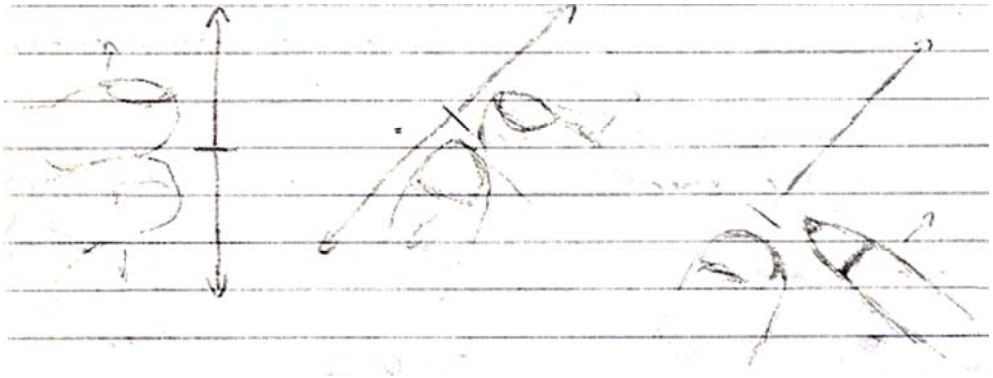


Figure 2: Concept - Pinching with two touch devices.

1.2 Connecting elements/Editing Connections

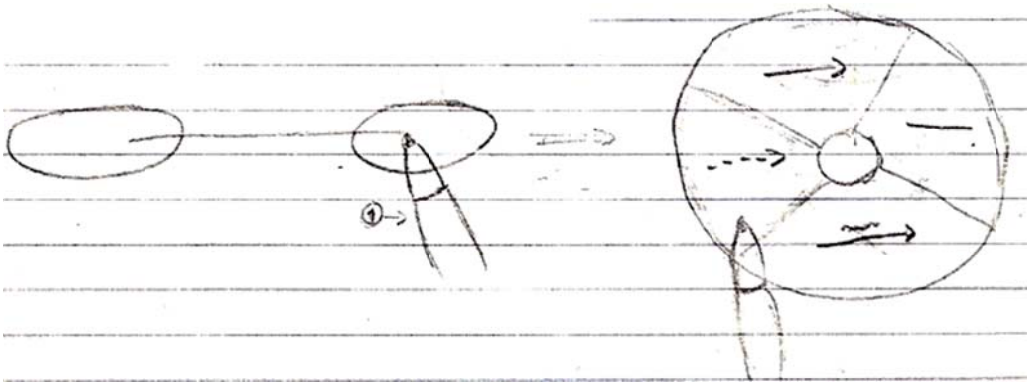


Figure 3: Concept - Adding connections.

To add a new connection the user could just draw a line from one element to the other and then a menu pops up with the different types of associations that can be created between those elements (Figure 3) or he could draw the whole connection as in a paper sketch.

To edit existing connections, that is bend and reroute the connection, the user could hold the connection with a finger as in Figure 4 and then with the pen or a second finger pul the connection, creating one editable vertex under the first finger and bending the connection with the pen or the second finger.

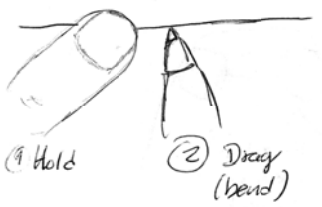


Figure 4: Concept - Editing connection pen and fingers at the same time

Or the user could hold it for an instant and let go, and then drag the connection using the pen or another finger as shown at the top of Figure 5. That would create two vertices in the places where the fingers where held and a the connection would be bent as the pen or third finger pull the connection.

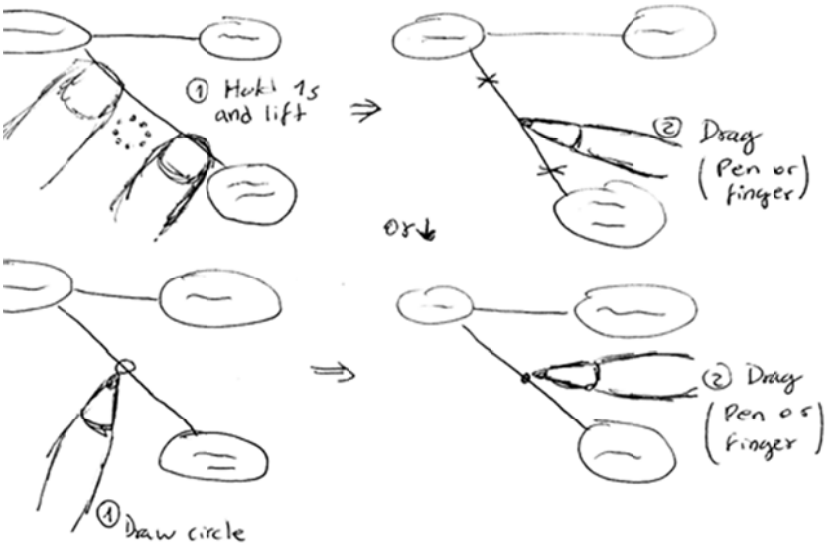


Figure 5: Concept - Edit connections with pen and fingers (top) or just pen (bottom).

Another alternative is the one presented in the bottom of Figure 5, using the pen the user could draw a small circle, representing a vertex and one editable vertex

would be created in the spot where of the circle, the user could then move the vertex using the pen or the finger.

Attaching or detaching a connection from an element to another could be done using the pen to cross out an end of a connection as in Figure 6, thereby detaching it and then drag that end point to another element.

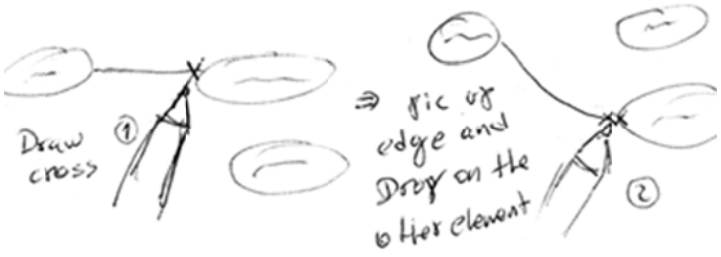


Figure 6: Concept - Attach/Detach connections with the pen.

Doing the opposite, attaching an element with an open connection that is, could be done by just simply drag the element to near the end point of that connection and the connection automatically attaching to the element, as shown in Figure 7.

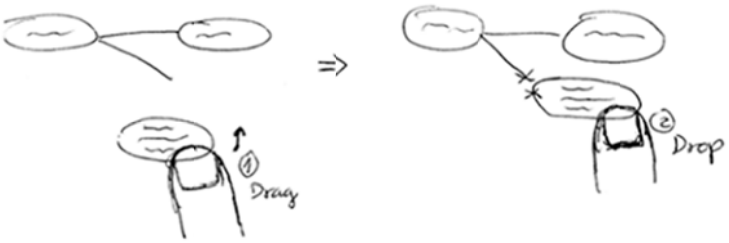


Figure 7: Concept – Attach elements to connections with the fingers.

Detaching a connection from an element could also be done using both the finger and the pen, in which the finger holds the element and the pen or another finger would then detach the connection by dragging it away from the element as sown in Figure 8.

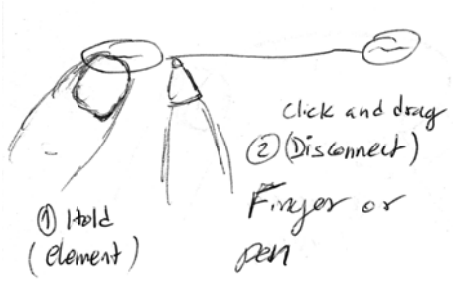


Figure 8: Concept - Detach connection form element with the pen.

1.3 Naming an element

To write the name of an element or to rename one, the user could double tap on the element as he would do with the mouse (double click) in a normal diagram tool, and then a box appears where the name can be inserted, either by virtual keyboard or by writing it with the pen as in Figure 9.

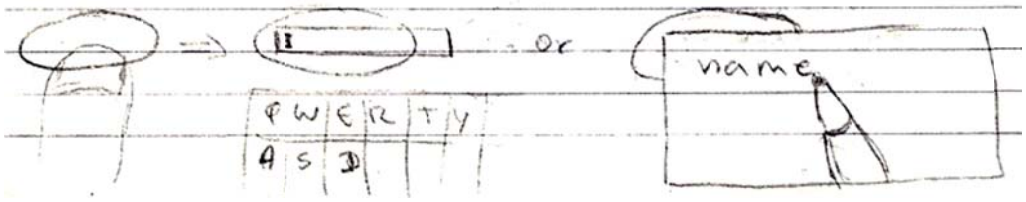


Figure 9: Concept - Name an element.

1.4 Adding elements

Simple sketch elements could be created by just drawing with the pen, as already seen with the connections, for instance a group could be created by simply sketching it around the desired elements as in Figure 10.

Figure 10: Concept - Adding a group using a pen gesture.

Complex elements can be added trough a toolbox or a widget panel, and it can be opened using a tab on the side of the screen that can be clicked or dragged out like in Figure 11. Another way could be like in Figure 12, the user could place tree fingers anywhere in the sketch and the menu slides out from underneath the fingers and the widget can be picked up with the pen or another finger and placed on the sketch. When picking up the widget from the panel, one tap would be used to add only one element to the sketch. Two taps would mean that the user could add multiple elements of the same.

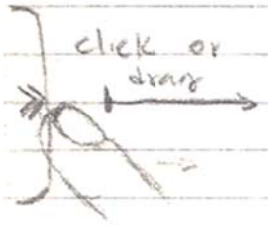


Figure 11: Concept - Side tab widget panel.

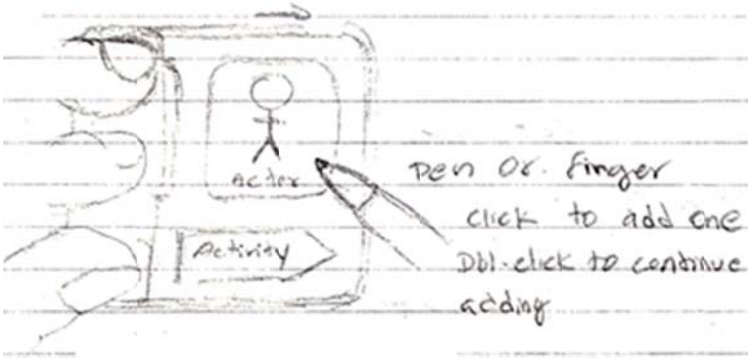


Figure 12: Concept - Floating widget panel.

Another way to bring up the toolbox could be by performing a gesture with the pen as in step 2 of Figure 13. On the left there is a concept for getting more whitespace on the sketch, the user would only have to drag the sketch to the side with the finger.

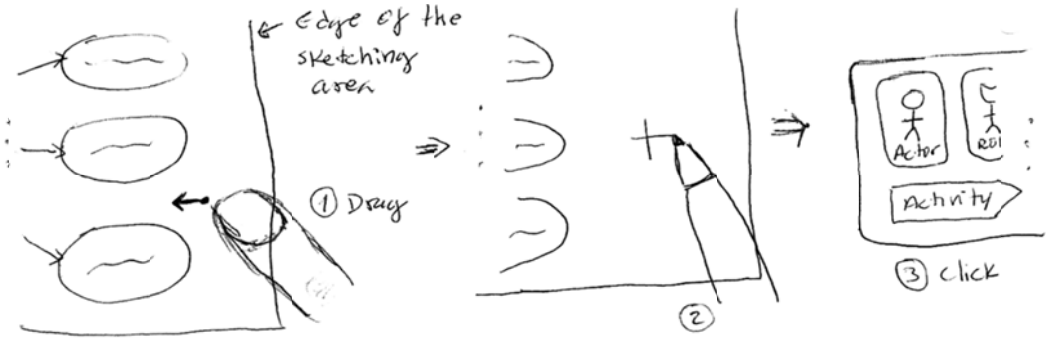


Figure 13: Concept – Getting more whitespace (left) and open widget panel with pen gesture (right).

1.5 Remove elements

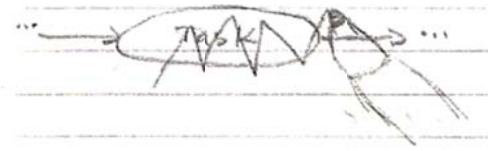


Figure 14: Concept - Removing an element using a pen gesture.

An element could be removes by making a scratching gesture over that element with the pen like in Figure 14.

Or the user could remove an element using the finger to drag an element to the edge of the sketching area or by flicking it away from its current passion as in Figure 15.

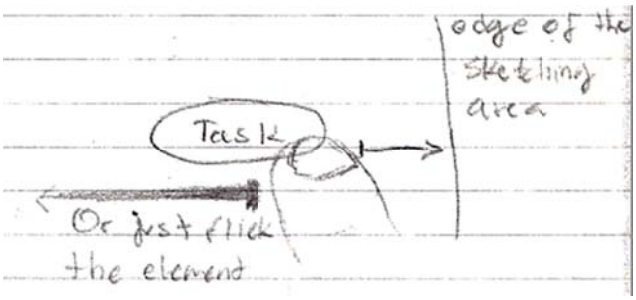


Figure 15: Concept - Removing an element with the finger.

To remove multiple elements the user could draw a rough border around the elements and without lifting the pen, cross out the whole thing as in Figure 16. Also finishing the gesture with scratching, like with one element, could also work.

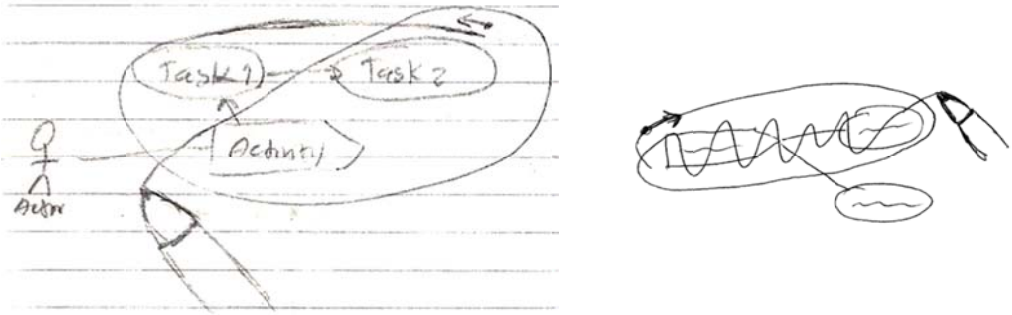


Figure 16: Concept - Remove multiple elements with a pen gesture.

Selected elements could be removed all in one, as they are selected all together; and if the pen used has a rubber on the back end of it, that rubber could be used to remove that selection and single elements as in Figure 17 and Figure 18.

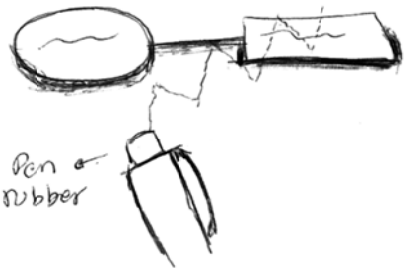


Figure 17: Concept - Removing selected elements using the back of the pen.

For elements that have attached connections or have other elements within themselves, before the system removes the element it could ask if the user wants to remove those attached elements and connections. If users do not respond in a pre-determined period of time then the option fades away and the default action is executed, only the target element is removed. This remove option could be one of two ways: just the remove as in Figure 18 left or remove/keep as in the right. Also the option would differ from groups to single elements, as for groups the elements within would

also be removed is the option was chosen. In the case of the single elements on the attached connections would get removed.

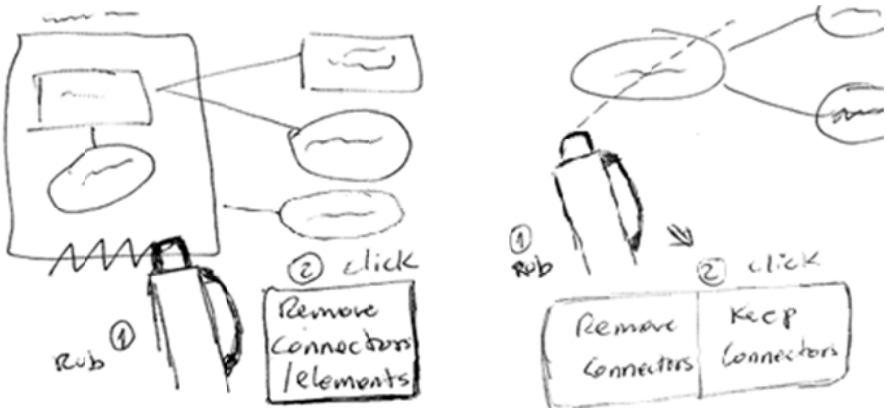


Figure 18: Concept - Removing a element using the back of the pen; Group remove option (left) and element remove option (right).

1.6 Selection

To select elements the user could tap on one using the finger or the pen. If the user had already a selection and wanted to add another element, he could hold the current selection with one finger and tap on another element with another finger or the pen as in Figure 19. A simpler solution could be to simply tap on elements to add them to a selection or to remove them from one.

To select multiple elements a lasso type interaction could be implemented, it could be done by putting a finger down and drawing a lasso around the elements, but starting and finishing near the finger, so it is not confused with a border. The finger could be held during the whole interaction as in Figure 20 left, or could be held for an instant and a ghost would then appear in place of it, from where the user could draw the lasso, like in Figure 20 right.

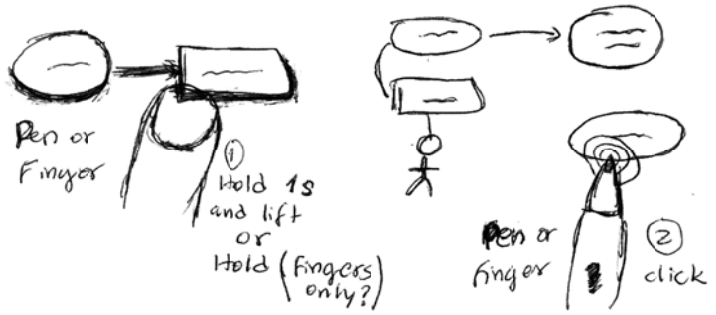


Figure 19: Concept - Adding elements to a selection with the finger and the pen.

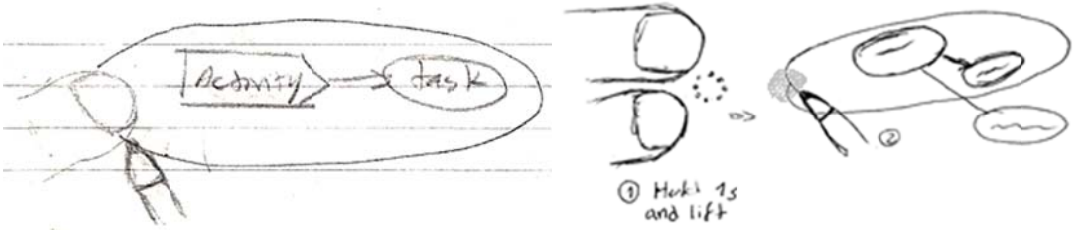


Figure 20: Concept - Selecting elements using the fingers and the pen at the same time (left) and not (right).

1.7 Copy an element

Figure 21: Concept - Copy an element.

Copying an element or a selection could be done using a similar interaction as the selection in Figure 19, except the user would have to tap or draw a make a plus gesture with the pen on whitespace to place a new copy of the held elements as shown in Figure 21.

1.8 Highlighting

To highlight an element the user could perform a gesture using the pen as he would in a paper sketch, for instance he could draw a contour around the element, or underline an element's name or emphasize the contour of the element as in Figure 22.

Figure 22: Concept - Highlight an element with a pen gesture.

1.9 Menu

A contextual menu could be designed that supports beginner users. It could be a radial menu that is better adapted to touch environment and it could be designed as in Figure 23, where there is an information icon near each option that would display an animation with the possible gestures or interactions to perform that option. To bring up that menu the user could hold with the finger or the pen for an instant or he could tap with two fingers on top of an element as in the bellow.

Possible commands would be Cut, Copy, Rotate, Move, Resize, Highlight, Rename, Remove and Select.

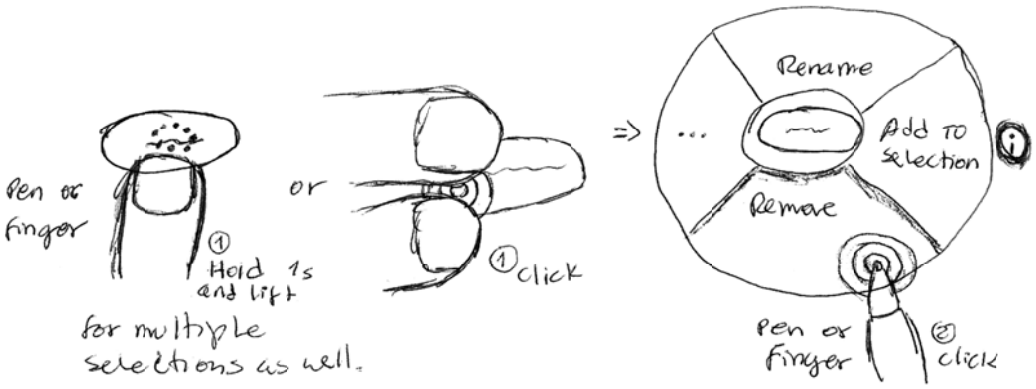


Figure 23: Concept - Bring up a menu with pen or fingers.

1.10 Rotate elements

Rotating elements can be done using the standard two finger rotation, either with one hand using the thumb and the index, or with two hands using one finger form each as in Figure 24.

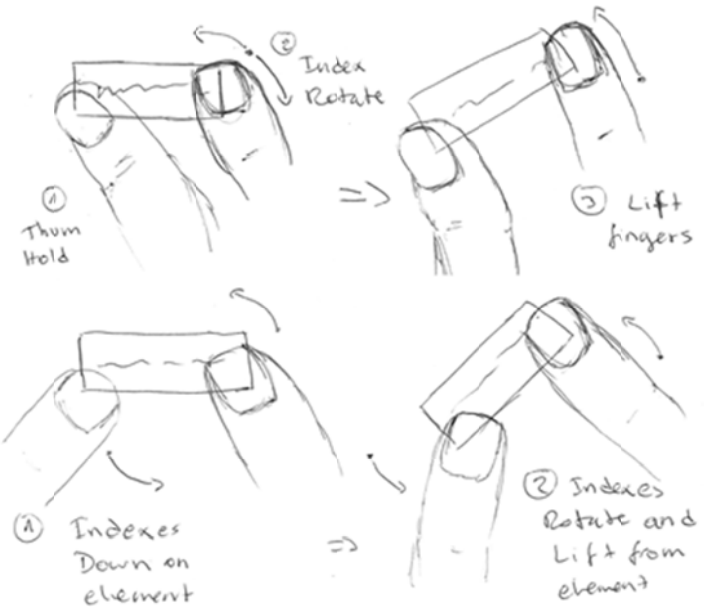


Figure 24: Concept - Rotate elements directly with the fingers.

A menu could be created to rotate the elements more precisely, where there would be predetermined degrees of rotation as in Figure 25 top. Another way of doing this would be using a widget that would be placed over the element when the rotate option would be selected form the menu, the widget would incrementally rotate the element when tapping and it would be precisely rotated when dragging the widget.

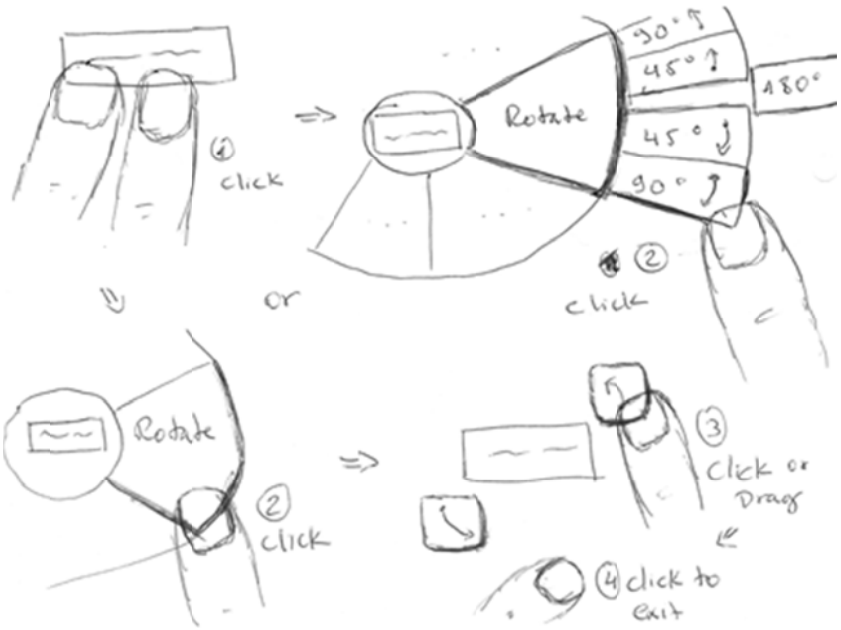


Figure 25: Concept - Rotate elements from the menu (top) or using a widget (bottom).

1.11 Resize elements

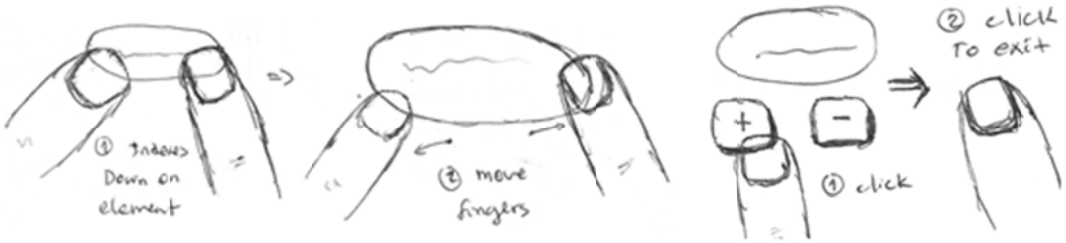


Figure 26: Concept - Resize elements directly with the fingers (left) and though a widget (right).

To resize elements the pinch interaction could be used, like with zoom, but it would have to be over the element, as in Figure 26 left. As with the rotation here a widget could be used for a more precise interaction, shown in Figure 26 right. Or the controls could be on the menu itself, as in Figure 27.

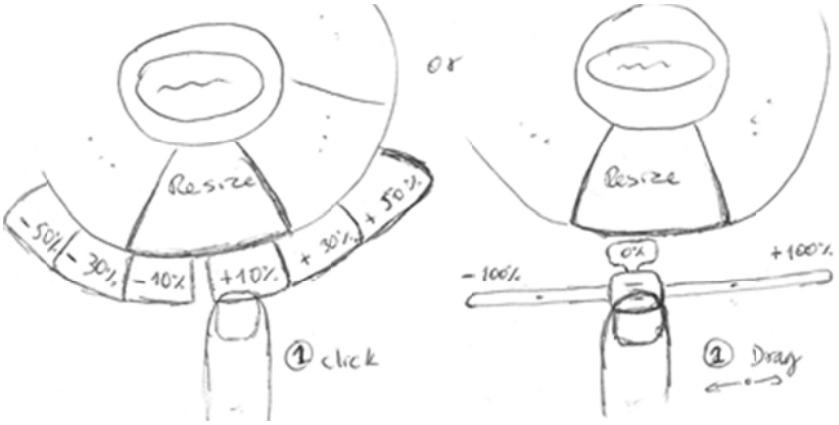
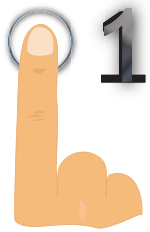
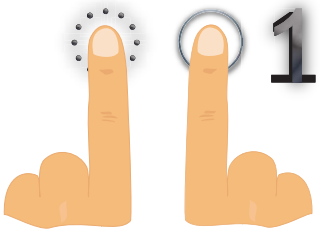
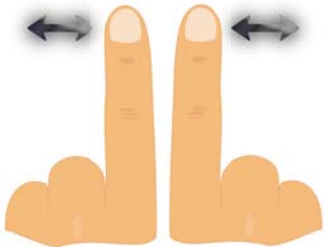
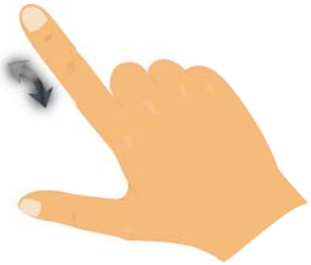
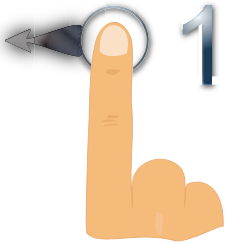
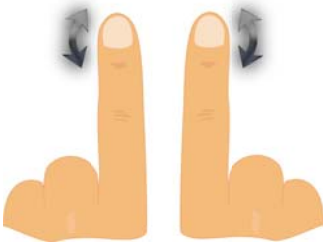
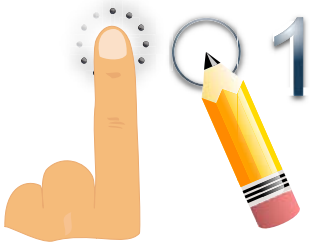
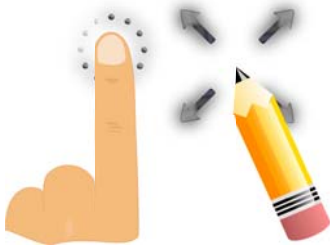
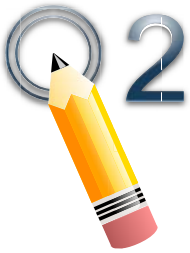
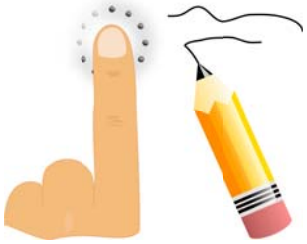


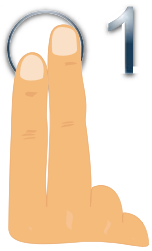
Figure 27: Concept - Resize element though a menu.

2 Affordances

This is a list of essential affordances that the multi-touch interface together with the pen provides.

Icon	Description	Icon	Description
	One finger tap		One finger hold and another finger tap (different hands)
	One finger double tap		Thumb drag and index finger drag in opposite ways (Same hand - Pinch)
	One finger flick		One finger drag and another finger drag in opposite ways (Different hands)
	One finger drag		Index finger rotates around thumb (Same hand)
	One finger tap and flick		One finger rotate and another finger rotate same way (Different hands)

	One finger tap and drag		One finger hold and pen tap
	One finger hold		One finger hold and pen double tap
	Pen tap		One finger hold and pen drag
	Pen double tap		One finger hold and pen draw
	Pen draw		Two finger hold and pen tap

	Pen erase		Two finger hold and pen double tap
	Pen tap and drag		Two finger hold and pen drag
	Two finger tap		Two finger hold and pen draw
	Two finger double tap		Three finger tap
	Two finger hold		Three finger hold

3 Action-Affordances Mapping

This is a list of affordances mapped to the actions elicited from the user studies, the conceptualizations and the affordances above. It also followed this design guideline:

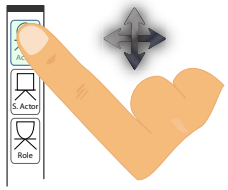
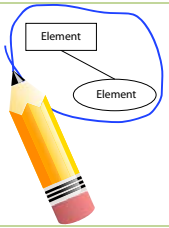
> *Because the target platform for this project is the small Tablet-PC/Slate with a 10 to 13' screen and the items and elements on the diagram are a bit small, a separation principle was established for the interaction devices, the Pen and the Fingers.*
>
> *The fingers are to be used for the rougher manipulations, for instance moving, resizing, rotating and selecting elements also zooming and panning the diagram, and such. The pen would be used for the more precise interactions and gestures such as drawing elements, connections and groups, renaming an element, editing a connection, highlighting an element, etc.*

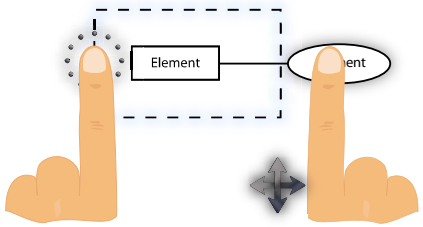
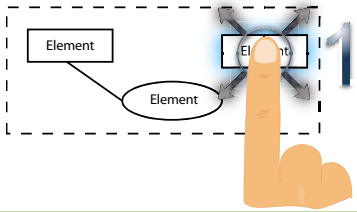
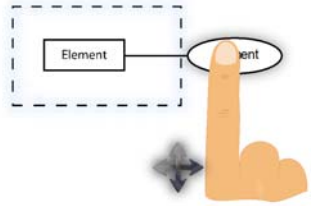
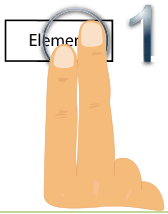
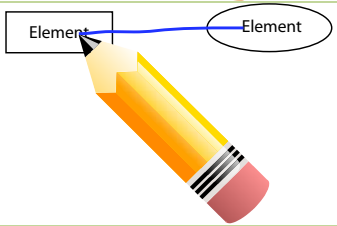
This list despite being the "finalized" concepts also acts as an inventory for the implemented interactions as it was updated throughout the project well into the implementation and user evaluations. In reality the first two stages of user implementations shaped some of the mappings seen in the list, as their purpose was to guide de design of the interactions and the interface.

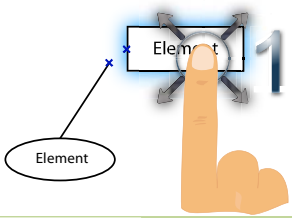
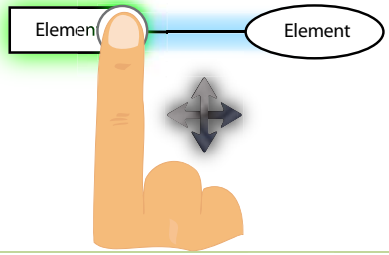
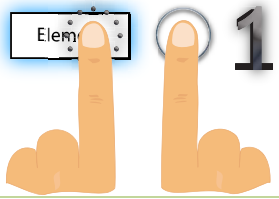
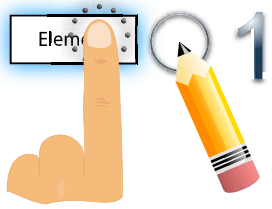
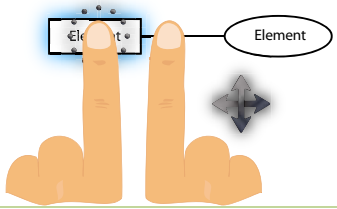
During those two initial stages of evaluation a review of the list was made to see if the mappings complied with the guideline above and as those stages where the design guide stages, the purpose of that review was to asses if any other interaction needed to be studied in the evaluations. The interactions studied on the evaluations either were deprecated if they showed negative results or kept if they were getting the evaluations approval.

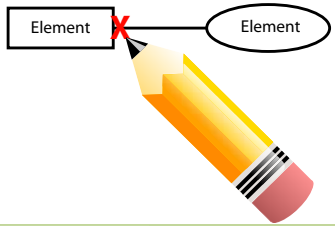
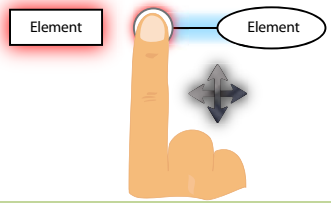
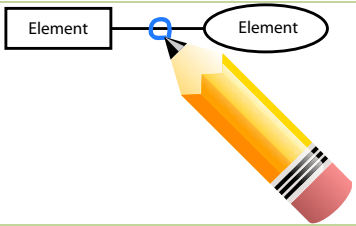
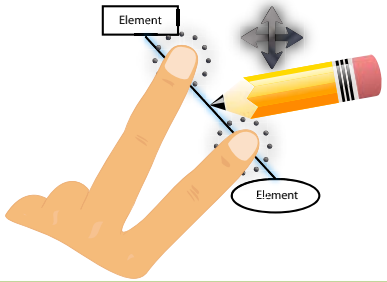
The actions are organized into groups of actions, as with the paper concepts. Each have a unique ID that is composed with initials from the action group that it belongs and the order in which it appears in that group. There is also an icon that represents the action, then the context in which is performed by the user, the outcome of the action and some observations about that action (issues, conflict with other actions, limitations, compliance with the design guideline, implementation status …).

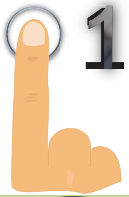
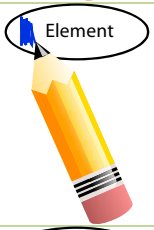
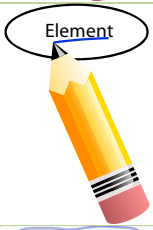
The **IDs** that are underlined mean that particular action was implemented in the prototype. The not underlined **IDs** mean that action was not implemented in the prototype because they were deprecated in benefit of another affordance or were not possible to implement. The actions marked with the italic ***IDs*** are not yet implemented.

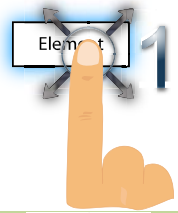
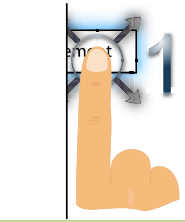
	ID	Affordances	Context	Result	Observations, Issues
Add an element	AE1		The element is dragged from the widget pallet, it does not have to be selected first	The element is added to the diagram in the position where the finger lifts off	Complies with the guide line, as it is the same as manipulating elements Implemented as a result from the user evaluations stage 1
	AE2		The tapped widget is locked in the toolbar	The element is added to the diagram in the position where the user tapped after choosing the element to add; the widget is released in the toolbar	Complies with the guide line, as it is the same as manipulating elements
	AE3			Several elements of the same type are added to the diagram, in the positions tapped by the user; the widget remains locked in the toolbar	
	AE4		Add an element by drawing it on the canvas with the pen	After the strokes are recognized they the corresponding element is added to the diagram on the same location as the strokes; then the strokes are deleted	Possible conflict with BO1, CN1, DC2, EC1 and HE1 Complies with the guideline, because is drawing new elements Will not be implemented as a result from the user evaluations stage 1
Borders/Groups	BO1		Around elements that already exist on the sketch	The border is created and the elements within it are now grouped	Possible conflict with EC1! Complies with the guideline, because is drawing new elements

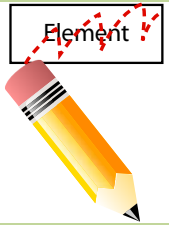
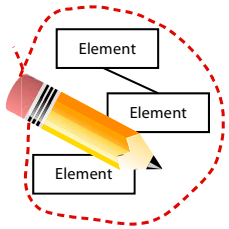
	ID	Affordances	Context	Result	Observations, Issues
	BO2		The holding finger is the first action and must be on top of a border (selected or not) that has at least one element; the element has to be fully dragged to the outside of the border	The moved element is removed from the group and afterwards the border adjusts to its content	Complies with the guideline, manipulating elements
	BO3		The element has to be moved and/or held to the top of an existing border	After the element is released, it gets added to the group and the border adjusts to its content	
	BO4		Remove an element from a border by just draging it out with one finger	The moved element is removed from the group and afterwards the border adjusts to its content	
Bring up a menu	BM1		On top of an element, selected or not	The menu appears around the element	Complies with the guideline, manipulating elements
Connect elements	CN1		The stoke has to start within an element and ends within another	A panel with possible connections appears	Complies with the guideline, manipulating elements

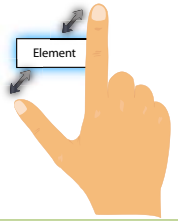
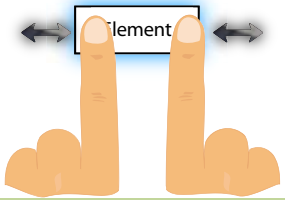
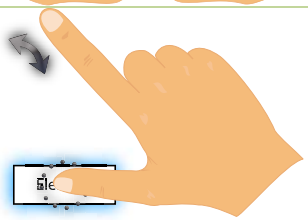
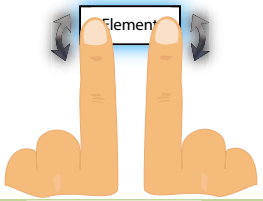
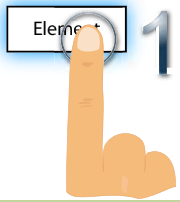
	ID	Affordances	Context	Result	Observations, Issues
	CN2		There has to be an open connection for the element to attach; the connection can also be manipulated in this form to attach to an element	The element is attached to the connection	Complies with the guideline, manipulating elements
	CN3		The connection has to be selected first for the handles to appear, then the handle has to be moved near or on top of the element		
Copy elements	CP1		The holding finger is the first action and must be on top of an element (selected or not) or a selection of multiple elements; the tap(s) must be on empty space	The selected element(s) are copied to the position(s) of the tap(s)	Possible conflict with SL3! Complies with the guideline, manipulating elements
	CP2				Possible conflict with SL4! Does not comply with the guideline, but might save time since there is no need to change from the pen to the finger Not implemented due to finger simultaneous with pen limitation
Detach a connection	*DC1*		The element is held first; the connection's edge must be dragged away from the element	The connection is separated from the element and stays open	Complies with the guideline, manipulating elements

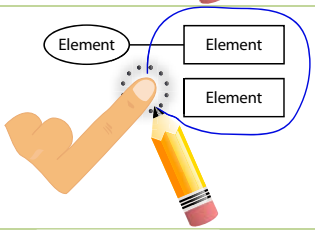
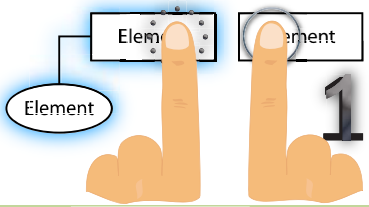
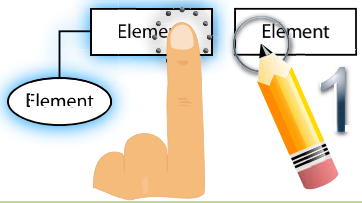
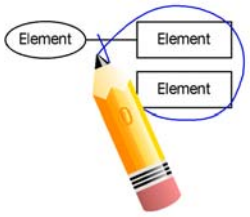
	ID	Affordances	Context	Result	Observations, Issues
	DC2		The stroke must be near the edge of the intended detachment side		Complies with the guideline, more precise interaction
	DC3		The connection has to be selected first for the handles to appear, then the handle has to be moved away from the element		Complies with the guideline, manipulating elements
Edit a connection	EC1		On top of a connection	A vertex is added to the connection and can be manipulated	Possible conflict with BO1! Complies with the guideline, precise interaction
	EC2		Both fingers have to be on top of a connection before the pen bends the connection	The connection is bent to the extent of the movement of the pen	Complies with the guideline, precise interaction Finger simultaneous with pen constraint

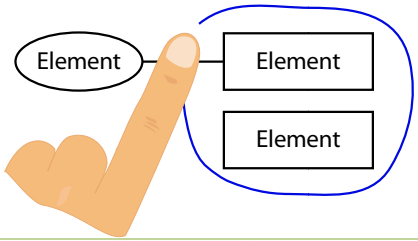
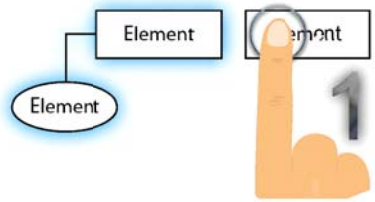
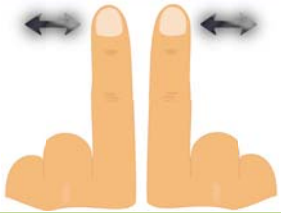
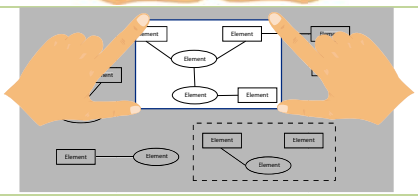
	ID	Affordances	Context	Result	Observations, Issues
Escape Action	ES1		The tap has to be on empty space	Escapes any action that is currently being performed (selection, editing a connection, an opened menu, editing a name, …)	Complies with the guideline, rougher interaction Will not be implemented because of a technology constraint, only two fingers are supported at the same time
	ES2				Complies with the guideline, rougher interaction
Highlight an element	HE1		The stroke must be performed inside the element, the contour also counts	The element gets highlighted or unlighted	Complies with the guideline, precise interaction
	HE2		The stroke has to be under the element's name or on top of it		Complies with the guideline, precise interaction Some connections might not have names, this would not work on those
	HE3		The stroke must go around only one element		Conflict with BO1 Complies with the guideline, precise interaction

	ID	Affordances	Context	Result	Observations, Issues
Move element(s)	MV1		The first tap is to select the element and it is not necessary if the element is already selected; groups of elements can also be moved by selecting and dragging them	The element is positioned where they were dropped	Complies with the guideline, manipulating elements Was deprecated as a result of the user evaluations stage 1
	MV2		The element is dragged to the edge of the sketching area; groups of elements can also be moved	The sketch is moved to the opposite direction until the element is released or the user backs off from the edge	Complies with the guideline, manipulating elements Not implemented due to lack of time
	MV3		Moving widhtout hanving to unlock the element first; the drag gesture has to be started on top of the element	The element is positioned where the finger is released	Complies with the guideline, manipulating elements Implemented as a result of the user evaluations stage 1
Name an element/ edit text	NE1		On top of any element, selected or not	A handwriting recognition canvas appears; the name can be written and added to the element	Complies with the guideline, manipulating elements
	NE2				Does not comply with the guideline, not a precise interaction, the double tapping gesture can equally be performed using the finger

	ID	Affordances	Context	Result	Observations, Issues
	NE3		The stroke has to be on top of the name		Conflict with HE1 and HE2 Complies with the guideline, precise interaction
Pan	PA1		Can be performed on any part of the sketch, apart from selected elements (that will move/remove the elements); can be performed with more than one finger	The sketch is moved over to the direction the user indicated	Possible conflict with MV1! Complies with the guideline, manipulating diagram
	PA2			The sketch moves to the direction indicated by the user and stops according to the speed used or when it reaches the edge	Complies with the guideline, manipulating diagram Not implemented, lack of time
Remove element(s)	RM1		The stroke has to be on top of at least one element, and has to be performed with the back end of the pen	The element(s) are removed from the sketch	The button to remove the items and connections attached is not implemented Complies with the guideline, precise interaction
	RM2		The stroke has to surround one or more elements		Does not comply with the guideline??? Maybe because it involves multiple elements but the user can precisely choose which to remove, a bit similar to SL5 Will not be implemented because lack of time, and the RM1 seems to be more efficient than expected

	ID	Affordances	Context	Result	Observations, Issues
Resize element(s)	RZ1		The element(s) has to be selected first	The element(s) is resized	Does not support the interaction outside the element Complies with the guideline, manipulating element
	RZ2				Complies with the guideline, manipulating element
Rotate element(s)	RT1		The element(s) has to be selected first; the thumb has to be placed before the index	The element(s) is rotated	Does not support the interaction outside the element Complies with the guideline, manipulating element
	RT2		The element(s) has to be selected first		Complies with the guideline, manipulating element
Select element(s)	SL1		On top of an element	The element is selected/unselected	Complies with the guideline, manipulating element

	ID	Affordances	Context	Result	Observations, Issues
	SL1a				Does not comply with the guideline, but the user might expect this functionality if SL5a is implemented
	SL2		The finger has to be placed first on an empty space; the stroke hast to surround one or more elements		Does not comply with the guideline, but similar to SL5 Not implemented because of a technology constraint, fingers and pen at the same time not supported
	SL3		The selected elements have to be held first; the taps can be on top of any element, selected or not	The elements are added/removed from the selection	Possible conflict with CP1! Complies with the guideline, manipulating element Not implemented because SL6 does not require the extra holding action thus being more efficient
	SL4				Finger and pen at the same time limitation Possible conflict with CP2! Does not comply with the guideline, but similar to CP2
	SL5		Selecting using the tip of the pen and pressing & holding the button at the same time; the lasso has to at least go around the elements center		Does not comply with the guideline, maybe because it involves multiple elements but the user can precisely and quickly choose which of them to select; Also users who used a tablet-pc before will expect to perform a lasso selection using the pen

	ID	Affordances	Context	Result	Observations, Issues
	SL5a		The user has to activate the selection tool in the toolbar; the lasso has to at least go around the elements center		Possible conflict with PA1 Complies with the guideline, manipulating elements But users might not expect this since it could be in direct conflict with PA1
	SL6		Adding and removing from selection by just tapping the element		Complies with the guideline, manipulating elements
Zoom	ZM1		Can be performed on any part of the sketch, apart from selected elements (that will resize them)	The view of the sketch is zoomed in/out	Complies with the guideline, manipulating diagram
	ZM2				
	ZM3		Can be performed on any part of the sketch, whether there are selected elements or not	The view of the sketch is zoomed out and the user can select the viewing area by manipulating the box	Cannot use more than 2 fingers Complies with the guideline, manipulating diagram

Appendix 3: User Tests and Evaluations

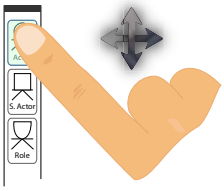
1 User tests – Stage 1

1.1 Functionalities/alternatives studied

In this evaluation three functionalities were studied and for each there were two alternative interactions. Here is a description of those alternatives and the hypothesis of the tests.

1.1.1 Add elements

1.1.1.1 *Alternatives*

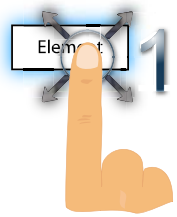
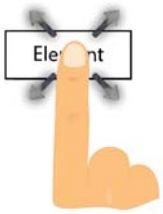
Adding elements by dragging them from the pallet	Adding elements by drawing them with the pen
 AE1*	 AE4*
Tested using Test 1	

1.1.1.2 *Hypothesis*

Adding elements by dragging them from the pallet is more effective than drawing them every time. The idea here is that the user will add elements faster and make fewer mistakes using the pallet, also that way the user does not have to remember exactly how the element looks like.

1.1.2 Move elements

1.1.2.1 *Alternatives*

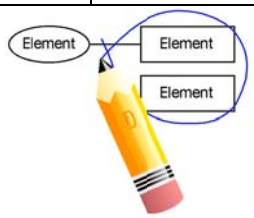
First tapping to unlock the element then move it, unselected elements are locked	Just drag with the finger to move an element, the elements are always unlocked
 MV1*	 MV3*
Tested using Test 2	Tested using Test 3

1.1.2.2 *Hypothesis*

Moving elements without the need to select/unlock them, is faster and more efficient than having to unlock them first. But having the elements constantly unlocked could allow for the user to commit more mistakes, such as unwillingly displacing an element when panning or doing some other operation that involves contact from the fingers with the area of an element that is not the target of that operation.

1.1.3 Select multiple elements with the pen

1.1.3.1 *Alternatives*

Selecting with the pen, using the pen's button as a press & hold	Selecting with the pen, using the pen's button as a toggle
 SL5*	
Tested using Test 4	Tested using Test 5

1.1.3.2 *Hypothesis*

Selecting by press & hold is the behavior the user expect, this comes from the behavior of the mouse in the existing diagraming tools which in general is, if one presses and holds the mouse button and moves the mouse it will select elements within the bounds of the selected area. So using the button to enter a mode of selection by just pressing it once and not having to hold it will not be as an effective interaction as the press & hold, because the user will be prone to commit more mistakes, for example starting to group elements but still being in selection mode or vice versa.

** - The full affordance description can be found in Appendix 2 – Concepts in section 3 Actions-Affordances Mapping.*

1.2 Task Sheet

Test 1

1. Add these elements as in the figure using the Panel

2. Draw the same elements a bit below the others using the stylus, with the precision that you think would be enough for the computer to recognize

Click NEW - **Test 2**

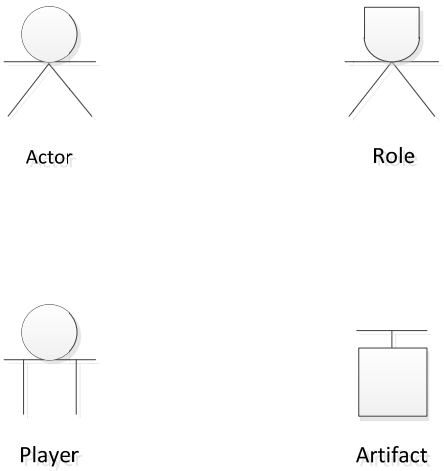
1. Positions 12 elements in 3 lines with more or less the same spacing as in the previous figure
2. Move an element the most to the right and another to the left
3. Move another upwards and another all the way down
4. Center the view again

Click NEW - **Test 3**

1. Repeat test 2

Click NEW - **Test 4**

1. Add four elements as in the figure below
2. Select the 2 left using the stylus and the side button
3. Connect the two top using the stylus
4. Select the lower 2 elements
5. Group the two right elements with the pen



Click NEW - **Test 5**

1. Repeat test 4

1.3 Tests and Results

In total were conducted five tests per user. Test 1 is related to functionality 1.1. Add elements, tests 2 and 3 are related to 1.2. Move elements and finally test 4 and 5 are about functionality 1.3. Select multiple elements with the pen. Next is a task sheet that was handed to the users that has the tasks that they have to do in order to complete the tests.

1.3.1 Test 1

This test had the objective of evaluating the addition of elements to the diagram. Therefor the user has to add elements from the toolbox and also draw those same elements using the pen. So here the two alternatives are then, adding elements using the finger with the toolbox and adding elements by drawing them with the pen. The hypothesis is that using the toolbox is faster and less prone to mistakes, as stated in hypothesis 1.1.1.2.

1.3.1.1 *Metrics*

- M1: How long the user takes to add all elements
    - Starts counting from the first element picked up to the last one dropped
    - Starts counting from the first stroke to the last

- M2: How long the user takes to add an element
    - This did not included the time users spent scrolling the pallet, in the case of dragging elements
- M3: How many times the user fails to start dragging the element from the pallet
- M4: How many times the user loses the element while dragging
- M5: How many mistakes the user commits while drawing
    - This means how many times the user erases parts of the sketch and redoes them
    - This also includes the retouches that the user could do to the drawing

1.3.1.2 *Results*

Metrics	M1 (s)		M2 (s)		M3	M4	M5
	Dragging	Drawing	Dragging	Drawing	Dragging	Dragging	Drawing
User 1	36,488	30,382	6,977	16,000	1	0	0
User 2	16,376	44,833	5,29	25,104	0	0	2
User 3	14,048	23,08	5,334	15,704	0	0	0
User 4	10,12	21,16	8,448	13,138	0	0	0

Drag vs. Draw

Metrics	M1 (s)	M2 (s)
Average	0,663	0,407

1.3.1.3 *Observations*

- As predicted users expect that dragging the finger down or up in the panel that it would scroll, even if it is on top of an element
- If the user is adding an element by drawing it he does not expect that the drawing disappears when he moves the pen away from the screen
    - That was a flaw in the test because the current implementation assumes that when the pen is no longer in range of the screen then there is no need to capture ink, and that should have not happened in this test
    - This is fine for a one stroke ink gesture; in case of multiple strokes this is no longer a viable assumption
- Some users used the pen instead of the finger to drag elements from the pallet
    - This option should have not been available during this test

1.3.1.4 *Conclusions*

Results show that for adding an element by dragging it from the pallet users took in average 41% the time that they took to draw an element; and taking into account the whole task of adding the three elements, when adding from the pellet users took only 66% the time to draw them.

More mistakes where expected, so the M3, M4 and M5 results are a bit surprising. This could be because the users where being careful on their drawings and because some users

dragged the elements from the pallet using the pen instead of the fingers. Even so the count of mistakes for drawing is still higher than dragging.

This indicates that the hypothesis is confirmed and dragging seems to be the better of the two options.

1.3.2 Tests 2 and 3

Test 3 is a repetition of test 2; the difference is that in test 2 the elements are locked and in test 3 they are not locked. The user has to drag four elements which are in the middle of other elements all the way to the four borders of the diagram. This is to see if during panning the users will accidently displace another element. And the hypothesis 1.1.2.2 is that users will make more mistakes when the elements are unlocked than when they are locked.

1.3.2.1 *Metrics:*

- M1: How many times the user tries to move an element when it is locked
- M2: How many times the user fails to tap the element before he succeeds on unlocking it
- M3: How many times the user moves an element by mistake

1.3.2.2 *Results*

	Test 2		Test 3
Metrics	M1	M2	M3
User 1	2	1	0
User 2	0	0	0
User 3	5	2	0
User 4	2	0	0
Total:	9	3	0

1.3.2.3 *Observations*

- Curiously even when the elements where locked users tended to hit the open spaces between them to pan through the diagram
- Some users expected when approaching the edge of the screen while moving an element that the view of the diagram would move in that direction, this was already expected
- One user said that he did not know how much more space there was on the diagram, so maybe there should be scroll bars on the zoom canvas
- The 1st and 3rd users did not notice that the elements where unlocked on test 3, so they always unlocked them first, but the 2nd and 4th users did noticed it

1.3.2.4 *Conclusions*

These results contradict the hypothesis, in the sense that users did more mistakes when the elements where locked than when they were unlocked, actually they did not do any mistake with elements unlocked. Users did not expect the elements to be locked, that could be because of the behavior of elements on other diagraming tools like Visio, in which elements are not locked.

The fact that there were no mistakes in test 3 alone does not indicate that the hypothesis fails. What leads to that conclusion is that users in test 2 still looked for empty space to preform panning actions, even after they knew that elements had to be unlocked to be moved.

So it seems that having elements unlocked will not lead the user to making the mistakes referenced in the hypothesis, thus invalidating it.

1.3.3 Tests 4 and 5

Test 5 is a repetition of test 4; the only difference is that in test 4 the pen's button operates as a press & hold and in test 5 the user only has to press the button once to enter selection or ink mode. These tests were designed so the user had to switch between selecting with the pen and other operations with the pen to maximize the use of the button. The hypothesis 1.1.3.2 is that using the button as a toggle is less effective and more propitious to mistakes than using the button as a press and hold.

1.3.3.1 *Metrics:*

- M1: How many times the user forgets that the pen is in a different mode than expected
- M2: How many times the user fails the selection in the middle of it
    - This means that the user did not complete the lasso around the targeted elements

1.3.3.2 *Results*

	Test 4		Test 5	
Metrics	M1	M2	M1	M2
User 1	1	2	6	0
User 2	0	0	3	0
User 3	x	x	x	x
User 4	0	1	6	0
Total:	1	3	15	0
	4		15	

1.3.3.3 *Observations*

- Users that are not accustomed to the pen take a long time to find the button
- Users do not find the cursors very clear, some more feedback is needed
- 1st user was so frustrated with the toggle that he gave up the last test
- User 2 had very difficulty finding the button on the pen and could not preform test 4 correctly, which made him very frustrated
- During user 3's test the camera's battery died, so test4 and 5 are not recorded

1.3.3.4 *Conclusions*

Results show that users committed about four times more mistakes in test 5 than in test 4, thus confirming the tests hypothesis and indicating that press & hold should be used instead.

During the execution of the tests users had difficulty finding the button on the pen, sometimes they had to even look at the pen to see where it was. This might be because the button is not salient enough and so it is not easily graspable.

That allied with users saying that the cursors are not feedback enough for the current mode, suggests that there is need for another way of perceiving which mode the pen is and to switch modes as well. For that purpose having controls in the top toolbar that provide feedback and allow changing modes should be sufficient to solve the problem.

2 User Tests – Stage 2

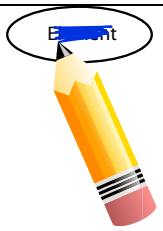
In this evaluation five functionalities were studied with the purpose of guiding the design and interaction validation. There were two alternative interactions for guiding the design and for validation only one interaction per functionality was tested. For the validation tests usability defects were gathered and not performance metrics as with the guiding tests.

Next is a description of the functionalities and the hypothesis for the tests. And then the tests themselves are presented.

2.1 Functionalities/alternatives studied

2.1.1 Name elements

2.1.1.1 *Alternatives*

Name elements, starting by Double-Tapping the element	Name elements, starting by Scratching the name of the element using the pen
 NE1*	 NE3*
Tested using Test 1	Tested using Test 2

2.1.1.2 *Hypothesis*

Typically in traditional diagraming applications, renaming elements can be performed by double-clicking the element with the mouse. The analogous interaction for a touch environment would be double-tapping with the finger or with the pen if such is supported. However according to the guideline double-tapping with the pen is not a precise interaction*, so it was not implemented for user testing.

Having this is mind double-tapping with the finger is the behavior the user would expect from the finger, and form the pen the user would expect scratching the name as in a paper sketch. But double-tapping an element should be faster than scratching the name of an element, primarily

because the target is bigger, it is the whole element. When scratching the user not only has to aim at the name of the element but, he has to perform the gesture correctly. So double-tapping is more efficient and will be prone to less user mistakes than scratching.

2.1.2 Highlight elements

2.1.2.1 *Alternatives*

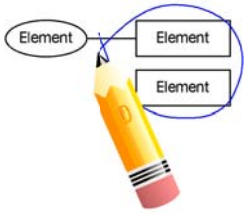
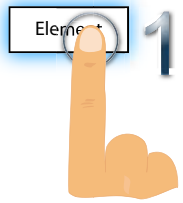
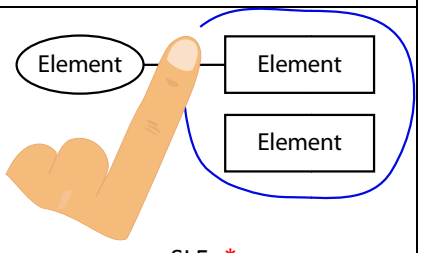
Highlight/Lowlight elements by contouring with the pen	Highlight/Lowlight elements by underlining the name with the pen
 HE3*	 HE2*
Tested using Test 3	Tested using Test 4

2.1.2.2 *Hypothesis*

From the user studies that were made, resulted that the actions performed to emphasize an element would be to add an extra contour or/and to underline the name, amongst other actions. But highlighting an element by underlining it should be less efficient than contouring it, because of the same reasons as in the last hypothesis – the target is smaller and the user is likely to make more mistakes – however contouring the element can be conflicting with grouping elements.

2.1.3 Select multiple elements with the pen

2.1.3.1 *Alternatives*

Selecting with the pen, drawing a lasso around elements using the pen's button and tapping on an element with the tip of the pen		Selecting with the pen, by drawing a lasso around the elements with the finger using the toolbar controls and tapping on an element also with the finger	
 SL1a*	 SL5*	 SL1*	 SL5a*
Tested using Test 5		Tested using Test 6	

2.1.3.2 *Hypothesis*

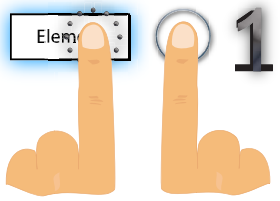
Lasso selecting with the pen is usual thing in tablet-pc environment, mainly because using the tip of the pen users can be more accurate with their selection than using the fingers which are bigger; also the first tablets did not support finger interaction as well as the latest models that use

the dual-digitizers. So using the pen for that action is as a legacy interaction and users will expect that, at least users that have past experience with tablet-pcs.

However selecting using the pen does not comply with the guideline* or was found to be a bit in a grey area between complying or not. So selecting with the finger was an alternative, but since the finger is bigger than the pen's tip users would more accurately select using the pen. But on the other hand selecting with the pen should take more time because users will lose time searching for the pen's button, and they should make more mistakes by accidentally releasing the button in mid selection.

2.1.4 Copy elements

2.1.4.1 *Interaction*

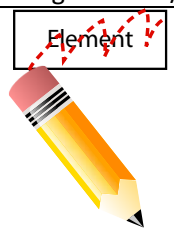
Copy and paste a selection, using one finger to hold the selection and another to tap the position to paste
 CP1*
Tested using Test 7

2.1.4.2 *Hypothesis*

Since this functionality was to be only validated there is no hypothesis.

2.1.5 Remove elements

2.1.5.1 *Interaction*

Remove elements using the back/rubber of the pen
 RM1*
Tested using Test 7

2.1.5.2 *Hypothesis*

Since this functionality was to be only validated there is no hypothesis.

** - The full affordance description can be found in Appendix 2 – Concepts in section 3 Actions-Affordances Mapping.*

2.2 Task Sheet

Test 1

1. Change the names of the elements for those who are here, write the name using the double-tapping with the pen

Test 2

1. Repetir o Teste 1 mas começa a mudar o nome rasurando com a caneta o nome actual

Test 3

There is a functionality, that is to make stand out or highlight an element

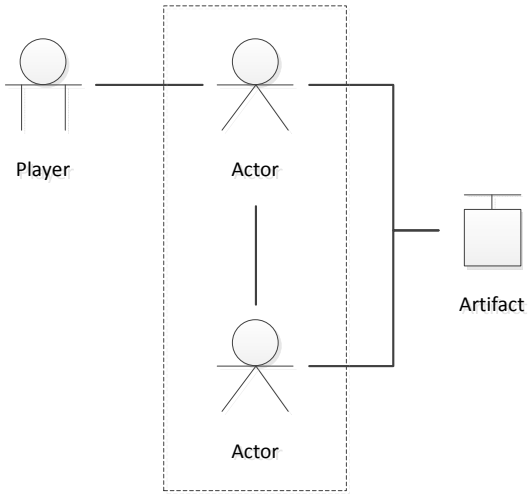
1. Highlight the Técnico de Suporte contouring it with the pen
2. Do the same with the connection Comunica
3. Now remove the highlight form Técnico de Suporte

Test 4

1. Repeat Test 3 but highlighting the elements by underlining their names with the pen

Test 5

It is possible to select or copy only one element or several elements



1. Using the pen and the button select the Player, the Actor beside it and the connection between them
2. Add the group to the selection using the pen

Test 6

1. Repeat test 5 but using the controls on the toolbar and the fingers for the selection

Test 7

1. Copy the selection and paste it using one finger to hold the selection and another to indicate where to paste
2. Remove the original group and all its elements and connections using the pen's rubber

2.3 Tests and Results

Next is a task sheet that was given to the users that contained the required tasks for the several tests.

2.3.1 Test 1 and Test 2

These tests have the objective of evaluating hypothesis 2.1.1.2, which is that double-tapping is more efficient and will be prone to less user mistakes than scratching. So they focus on changing the names of the components. In Test 1 the user has to change the names starting by double-tapping the components. In Test 2 the user starts changing the names by scratching the current name of components.

The tests also have the objective of validating the Ink Recognition panel for writing the name of element. Afterwards the user has to use the ink recognition panel to write the new name using the pen and insert it into the component.

2.3.1.1 *Metrics*

- M1: How many times the user fails to double tap a diagram component
- M2: How many times the user fails to scratch the name of a diagram component
- M3: How many usability defects found with the Ink Recognizer

2.3.1.2 *Results*

	Test 1	Test 2
Metrics	M1	M2
User 1	0	2
User 2	3	7
Total:	3	9

M3 - Usability Defects			
Defect #	**Title**	Ink Recognizer: Recognize button	
1	**Feature**	Button	**Notes:**
	Problem	Interaction Design: confusing, uninformative	
	Usability Principle	Simplicity/efficiency	
	Severity (1 = nominal, 2 = minor, 3 = major, 4 = critical, ? = evaluate)	2	
Defect #	**Title**	Ink Recognizer: Element label/textbox size	
2	**Feature**	Text	**Notes:**
	Problem	Presentation Design: confusing, uninformative	Text too big for the label/textbox, inside the

	Usability Principle	Visibility/availability, feedback	diagram elements
	Severity (1 = nominal, 2 = minor, 3 = major, 4 = critical, ? = evaluate)	2	
Defect #	**Title**	Ink Recognizer: tapping	
3	**Feature**	Panel	**Notes:**
	Problem	Interaction Design: Unexpected	User expected the name to be inserted, but it canceled
	Usability Principle	Simplicity/efficiency	
	Severity (1 = nominal, 2 = minor, 3 = major, 4 = critical, ? = evaluate)	4	
Defect #	**Title**	Ink Recognizer: ruler line	
4	**Feature**	Graphic, panel	**Notes:**
	Problem	Presentation Design: function hidden	User was wondering if the space for writing is only over the ruler line
	Usability Principle	Feedback, structure/organization	
	Severity (1 = nominal, 2 = minor, 3 = major, 4 = critical, ? = evaluate)	3	
Defect #	**Title**	Ink Recognizer: Recognition	
5	**Feature**	Function, text	**Notes:**
	Problem	Interaction Design: confusing, hidden behavior, slow, unexpected	If the user backs the pen away from the screen in the middle of writing the name it will start recognizing the ink and slow down the interface considerably
	Usability Principle	Feedback, structure/organization	
	Severity (1 = nominal, 2 = minor, 3 = major, 4 = critical, ? = evaluate)	1	
Defect #	**Title**	Ink Recognizer: Element textbox visibility	
6	**Feature**	Function, text, view	**Notes:**
	Problem	Presentation Design: confusing, function missing, nonstandard	It sends the affordance of text editing on the textbox
	Usability Principle	Visibility/availability, feedback, reuse/consistency	
	Severity (1 = nominal, 2 = minor, 3 = major, 4 = critical, ? = evaluate)	3	

2.3.1.3 *Observations*

Note:
The first user tests were not recorded by mistake, but notes were taken about them. So it will be designated as User 0.

User 0

- Initially the user tried to write the element's name on top of the current name before even thinking about giving a double tap or any other action to start changing the name
- The user's first attempt to scratch an element was to make a dash over the name as the highlight of the test 4
- To scratch the elements the user failed a few times because the strokes were not horizontal or because he did not make the number of zig-zags needed for the gesture to be recognized
- When writing the name of the element the user tried to write it the clearest and on top of the text box that appears when an element is editing its name
- The user expected that a few seconds after he finished writing the name of the element, it would be inserted automatically
- The user did not understood what was the recognize button for

User 1

- The user expected that tapping after writing the name of the element that would insert it
- Also he tried to start editing the name of the second element wright after writing the name of the first one, without inserting it

User 2

- User did not know what the recognizer button was for, so he was hesitant in clicking it

2.3.1.4 *Conclusions*

Results show that in test 2 users made triple the mistakes they made in test 1, this confirms the hypothesis and as expected double tapping an element is more efficient and is susceptible to less mistakes than scratching an element.

Now these results may be influenced by the facts that the scratch stroke has to be horizontal for it to be recognized and the users had difficulty understanding the gesture as they initially thought that it was a simple dash as in highlighting. Also all users had little to no past experience with tablet-pcs, however with the mouse is a different scenario as they all use it every day.

Regarding scratching the name there still is the fact that in the future with the implementation of several types of connections, and since some of them might have no name when created but the user could add a name subsequently, the implemented gesture should not be in function of the current name.

During the tests was observed that one user tried to write the name of the elements on top of the text box of the element, this could be because the textbox sends out the wrong affordance to the user of editing the text directly on the textbox, as it is a standard behavior. Also one user tried to start edition of another element immediately after writing down the name of one element. Putting these too observations together it seems that the user had perceived the affordance of being able to manipulate the elements on the diagram when writing the name.

Regarding defect 3, one user expected that tapping would insert the new name; however tapping in a clear space is the default escape action, and changing the tap to insert the new name would create inconsistency.

2.3.2 Tests 3 and 4

Tests 3 and 4 relate to the hypothesis 2.1.2.2, highlighting an element by underlining it should be less efficient than contouring it. So they are only focused on highlighting and lowlighting components. In test 3 the user has to highlight by contouring the components with the pen. On test 4 he has to underline the name of the elements to highlight the elements.

2.3.2.1 *Metrics*

- M1: How many times the user fails to contour the component
- M2: How many times the user fails to underline the component's name

2.3.2.2 *Results*

	Test 3	Test 4
Metrics	M1	M2
User 1	0	1
User 2	0	1
Total:	0	2

2.3.2.3 *Observations*

User 0

- Contouring the elements was the first guess that the user gave when asked what would he have done on a paper sketch to make an element stand out, the following was to make the letters bold
- When trying to underline the name of the elements, the user was unable to highlight the element, sometimes because he was doing the gesture more slowly, others because the line was not completely horizontally and it fell out of the element's range

User 1

- User says that contouring the elements again to remove the highlighting seems weird
- User says that the effect of highlighted is to similar to the effect of the element being selected

User 2

- When underling, the strokes were not to accurate, but still they were recognized correctly for the most part
- To highlight two elements at once the user tried contouring the two of them at once and a group was created, and in test 5 he already knew the difference between the two interactions.

2.3.2.4 *Conclusions*

The results collected in these tests show that users committed no mistakes in contouring the elements and only two when underling the element, thus confirming the hypothesis. However the difference is not that big and once users picked up the underling gesture they said it made sense. But then again if an element has no name it cannot be highlighted using the underline.

The mistakes committed while underlining might be related to how the gesture is implemented, which is using the gestures Left and Right from the Windows InkCanvas. The problem is that they do not allow for much noise on the stroke which puts constraints on their use.

One user after making the tests tried highlighting multiple elements by contouring them and ended up grouping them, that was a mistake he made only once and learned grouping from it and started experimenting creating groups with other elements and highlighting other elements. Since creating groups was not a functionality tested at this stage, the user did not know that it was possible to create groups until then. This means that even if the two gestures are in conflict, they are sufficiently distinct that users can understand the difference and start using the gestures immediately. As some of the interactions are made through gestures which have to be discoverable by the users, the fact that the user was experimenting with the interface is a benefit because means the interface is exploarble.

Regarding the connections, instead of users underlining the name they tried to overload the connection itself and some said that contouring a connection did not make much sense. Maybe it is because the result of contouring an element is adding an actual contour to the visual of the element and in a connection it is a different outcome since no contour is added, the effects are only applied to the current line. This might indicate that a different gesture should be used to highlight connections than the one used to highlight elements, maybe something more related to the end result of the highlight of the connection.

These are somewhat confusing results but they point to some aspects that have not been forethought, for instance the different approaches users make when highlighting connections vs. elements. And during the initial user studies when asked what they would do to highlight an element, users responded that they would contour or underline the element, so both gestures should be kept.

2.3.3 Tests 5 and 6

Tests 5 and 6 put to trial the hypothesis 2.1.3.2, that states that users would more accurately select using the pen, but on the other hand using the pen should take more time and commit more mistakes because of the button. As such the tests concentrate on selecting elements. On test 5 users have to select the elements using the pen to lasso around them and tap on an element to add it to the selection. Whether in test 6 the user has to use the finger to perform the selections.

2.3.3.1 *Metrics*

- M1: How many times the user fails the lasso selection in the middle of it
  - This means that the user did not complete the lasso around the targeted elements
  - Not that he thought it was in selection mode and it was not
- M2: How many times the user fails to tap the element to select it
  - This counts only the taps that were actually on top of element but for some reason did not select them

2.3.3.2 *Results*

	Test 5		Test 6	
Metrics	**M1**	**M2**	**M1**	**M2**
User 1	0	0	0	0
User 2	2	0	0	0
Total:	2	0	0	0
	2		0	

2.3.3.3 *Observations*

User 0

- The user did not expect that could tap with the stylus
- To select with lasso the user expected to add to the existing selection

- Again, like the button on the pen, the user did not expected that the selection on the toolbar to be active after making a successful selection with the finger

User 1

- No observations were made for this user

User 2

- User did not consider that tapping with the pen he could select an element, he said because of the lasso that it was too different to start tapping
- The user expected after having something already selected, that the lasso selection would add to the selection and not create a new one (but this is not the standard behavior???)
- When selecting with the finger the user at first tried without using the toolbar, but at the second try used it

2.3.3.4 *Conclusions*

Results are very similar on both tests, but as expected one user had a bit more difficulty on test 5 lassoing with the pen. However the lasso selection with the pen is a legacy interaction and it saves time when the user is interacting using the pen, as he has not to switch to the fingers to do the selection. So lasso selection with the pen should be maintained.

Tapping with the pen was an unexpected behavior by the users. It did not seem natural in the context of the diagram, since they could already tap using the fingers. However users did not commit any mistakes wen performing the taps and for the same reasons as with pen lasso selection it should be kept, those are because tapping is a legacy interaction used in tablets and if the user is interacting using the pen he does not have to switch to the finger to add an element to the selection.

About the fingers interactions, users did not commit any mistakes in the interaction. However they expected the controls on the toolbar not to function as full toggle buttons, in the sense that when they finished a selection they expected the control to be deactivated. It that would be the case, then the interface would be introducing hidden behavior and inconsistency as it would not be consistent with the other controls for the ink and rubber.

With the lasso selection either in the case of fingers or pen, it was observed during the test that users expected the elements selected to be added to an existing selection and not create a new selection. Now this is a surprising user expectation, since the standard behavior is that a lasso tool creates new selections. This aspect would need more study to make a more informed decision.

2.3.4 Test 7

The test has the objective of validating functionalities 2.1.4 Copy elements and 2.1.5 Remove elements. To that purpose the user has to copy and paste the selection made on test 6. Then the user has to erase the group, all elements within and all the connections related to them from the original components.

2.3.4.1 *Metrics*

- M1: How many usability defects found with functionality 1.4

- M2: How many usability defects found with functionality 1.5

2.3.4.2 *Results*

M1 - Usability Defects			
Defect #	**Title**	Copy: held item effects	
1	**Feature**	Detail, graphic	**Notes:**
	Problem	Presentation function missing, inconsistent, uninformative	The activation effect is only shown on the item that is being held and not on the rest of the selection
	Usability Principle	Feedback, reuse/consistency	
	Severity (1 = nominal, 2 = minor, 3 = major, 4 = critical, ? = evaluate)	2	
Defect #	**Title**	Copy: Selected elements	
2	**Feature**	Color, detail, graphic	**Notes:**
	Problem	Interaction Design: inconsistent, nonstandard, unexpected	Only the element under the finger at the destination gets selected
	Usability Principle	Feedback, structure/organization, reuse/consistency	
	Severity (1 = nominal, 2 = minor, 3 = major, 4 = critical, ? = evaluate)	3	

2.3.4.3 *Observations*

User 0

- To copy the elements, as the user was not accustomed to use touch environments but the more conventional mouse and keyboard, he was not understanding the hold & tap technique, he expected to use the toolbar buttons
- To remove the element the user tried:
  - Initially to use the toolbar rubber button and the finger
  - Selecting them and then click on the button clear that is on the toolbar

User 1

- The user forgets that the selection is still activated on the toolbar from the last test

User 2

- The user forgets that the selection is still activated on the toolbar from the last test

2.3.4.4 *Conclusions*

When removing elements using the pen, there was no usability defects found. Furthermore it was observed that the performance of the interaction was better than expected, even with the option of removing elements and connections that are attached to a group not yet implemented. The performance was good enough to even consider not having that option implemented as conceptualized.

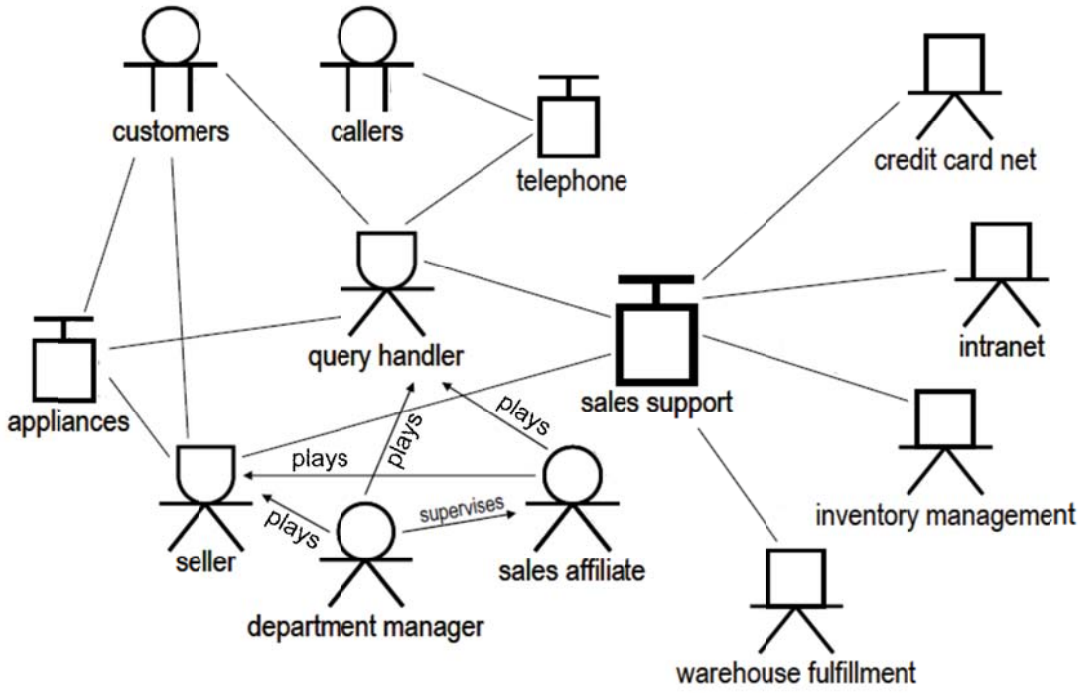
With copy of elements two usability defects were found, the defect 1 is about the visual effects of holding an element which is they get elevated. The defect is that only the element under the finger has the effect, where the full selection should have the effect. Defect 2 is about the elements that are pasted not being all selected as they should. This is a more serious defect because if the user wants to adjust the position of the elements, he has to select them again thus

not being much efficient. The first defect is not as important as the selected elements all get copied despite not all having the held effect.

3 User Tests – Stage 3

This stage is the final stage and was comprised of one major test which was the creation of a diagram in three different environments. Afterwards the user answered a small questionnaire comprised of questions about the user's existing experience with diagram creation and questions about the creation of the diagrams in the tools.

3.1 Diagram



As the same diagram was asked to be created three times there could be issues of the users learning the diagram and become faster with each test. For it not to be an issue that impacts heavily on the outcome of the test, the users were asked to start the diagram from a different part for each of the three tools. Moreover the tools where so much different from each other that the user could not be affect by doing the same actions and eventually learning them and getting faster doing them.

3.2 Questionnaire

After the user completed the diagrams he was asked to complete the following questionnaire. His answers are in black dots on the numbered columns.

0 = Lowest; 4 = Highest

	0	1	2	3	4
1. General experience with diagrams	○	○	○	●	○
2. Experience with HAM –Human Activity Modeling	○	●	○	○	○
3. General easiness creating the paper sketch	○	○	●	○	○
4. General easiness creating the diagram on MetaSketch	○	○	○	●	○
5. General easiness creating the diagram in the prototype	○	○	○	○	●

The following questions are only related with the prototype:

Evaluate the experience you had using the prototype relating:	0	1	2	3	4
6. Adding elements	○	○	○	●	○
7. Selecting elements	○	○	○	○	●
8. Positioning and adjusting elements' positions	○	○	○	○	●
9. Changing elements' name	○	○	○	●	○
10. Navigating through the diagram	○	○	○	○	●
11. General use of the pen	○	○	○	●	○
12. General use of touch	○	○	○	●	○

The user said that he did not give 4 in the questions 6 and 12 because of the touch responsiveness of the touch-screen. Regarding question 9, it was because he did not like to write with the pen on the tablet. For question 11 he said that it was the barrel button in the pen that he accidently pressed.

3.3 Metrics

- M1: How long the user takes to complete the diagram in paper
  - Did not include the time to add the names of the connections
- M2: How long the user takes to complete the diagram in MetaSketch
  - Did not include the time to add the names of the connections
- M3: How long the user takes to complete the diagram in the prototype
  - Did not include the time to add the names of the connections

- M4: How many mistakes the user commits while drawing in paper, including:
    - Includes the retouches that the user could do to the drawing
    - Scratching elements and redrawing them on another part
    - Having to put elements on another part of the sketch because the space was miscalculated
    - And other interaction mistakes like so
- M5: How many mistakes the user commits while drawing in MetaSketch, including:
    - Accidentally moving undesired elements
    - Failing to click an element to select it
    - Failing to double click an element to start editing the name
    - Missing the targets for adding a connection
    - Dropping an element while moving it through the diagram
    - Misspelling a name
    - And other interaction mistakes like so
- M6: How many mistakes the user commits while drawing in the prototype, including:
    - Retouches that the user could do when drawing/writing with the pen
    - Accidentally moving undesired elements
    - Failing to tap the element to select it
    - Failing to double tap an element to start editing the name
    - Missing the targets for adding a connection
    - Dropping an element while adding it to the diagram
    - Dropping an element while moving it through the diagram
    - Misspelling a name
    - And other interaction mistakes like so

3.4 Results

Metrics	M1 (M:S)	M2 (M:S)	M3 (M:S)	M4 #	M5 #	M6 #
User 1	03:37	04:46	06:43	3	4	6

3.5 Observations

- The performance of the prototype started to deteriorate when many elements were added, the same was not true with Metasketch
- The user clearly was not costumed to use tablet handwriting recognition, because in the beginning he wrote very much slower than further into the test, by the end, when editing the names of the last few elements the user wrote them very much faster than in the beginning.
- The user says the HAM elements look all alike.

3.6 Conclusions

Results show that the prototype took more time than the paper and MetaSketch tests and also that the user committed more mistakes. However several factors contributed for this to happen, namely the performance of the prototype seemed to deteriorate considerably when using the screen capture software which affected the touch responsiveness and the ink

recognition timings. This performance drop only worsened when more elements were added and the diagram grew in size, however the same was not noticed with the MetaSketch. Those factors both contributed to a slower competition time and for the user to commit more mistakes as things were moving and responding slower that he expected.

Additionally the user said that he did not like to write on the Tablet-PC and was not too fond of the pen either, especially because of the barrel button that was unwillingly pressed.

Then after the user responded to the questionnaire and said he does not have much experience with human activity modeling, he commented that the elements were somehow confusing because they seemed all alike. It is true that there are some similarities within the elements from the participation map, as most have some sort of legs, arms and a head; and this can be confusing to someone who was not trained or have experience with the language. Maybe the differences between the widgets could be emphasized somehow, using colors or making the arms and legs longer.

About the questions about the user's evaluation of his experience with the tool, the user responded with values of 3 and 4, which means that in the user's opinion there are still some aspects that could be improved, but overall is good. When questioned of those aspects the user responded that they had to do with the pen itself and the button. That leads to the conclusion that this button, despite being useful gets in the way of the users and maybe should be deactivated.