



A Methodological Framework for Multi-Fidelity Sketching of User Interfaces

By Adrien Coyette

A dissertation submitted in fulfillment of the requirements for
the degree of

Doctor of Philosophy in
Management Sciences

of the Université catholique de Louvain

Committee in charge:

Prof. Jean Vanderdonckt, Advisor

Prof. Manuel Kolp, Advisor

Prof. Stéphane Faulkner, FUNDP, Examiner

Prof. Laurence Nigay, Université J. Fourier, Reader

Prof. Pierre Leclercq, Université de Liège, Reader

Autumn 2007

Table of content

| | |
|---|-----------|
| TABLE OF CONTENT | 5 |
| CHAPTER 1 INTRODUCTION | 13 |
| 1.1 THESIS | 20 |
| 1.1.1 Thesis statement | 20 |
| 1.1.2 Validation | 21 |
| 1.1.3 Scope | 21 |
| 1.2 READING MAP | 22 |
| CHAPTER 2 STATE OF THE ART | 25 |
| 2.1 PROTOTYPING | 26 |
| 2.1.1 Levels of fidelity | 27 |
| 2.1.2 Prototyping development paths | 29 |
| 2.1.3 Prototyping types | 33 |
| 2.1.4 Scope of prototyping | 36 |
| 2.1.5 Prototype executability | 36 |
| 2.2 ANALYSIS GRID | 38 |
| 2.3 CLASSICAL APPROACHES | 40 |
| 2.3.1 The Paper prototyping | 40 |
| 2.3.2 Tiny fingers prototyping | 43 |
| 2.4 UI SKETCHING APPLICATIONS | 45 |
| 2.4.1 Silk | 45 |
| 2.4.2 Denim | 48 |
| 2.4.3 Gabbeh | 50 |
| 2.4.4 CrossWeaver | 52 |
| 2.4.5 JavaSketchIt | 54 |
| 2.4.6 FreeForm 2 | 57 |
| 2.4.7 Inkkit | 60 |
| 2.4.8 GUI Design Studio | 63 |
| 2.4.9 Visio | 65 |
| 2.4.10 stpBA Storyboarding | 67 |
| 2.4.11 MockUpScreens | 69 |
| 2.4.12 Axure RP | 72 |
| 2.4.13 GUILayout | 74 |
| 2.4.14 EasyPrototype | 76 |
| 2.5 OTHER SKETCH BASED APPLICATIONS | 79 |

| | | |
|--|--|------------|
| 2.5.1 | <i>EsQUIse</i> | 79 |
| 2.5.2 | <i>SketchRead</i> | 80 |
| 2.6 | SUMMARY | 81 |
| 2.7 | REQUIREMENTS FOR SKETCHIXML | 82 |
| CHAPTER 3 SKETCHIXML DEVELOPMENT | | 87 |
| 3.1 | DEVELOPING USER INTERFACES FOR MULTIPLE CONTEXTS OF USE | 87 |
| 3.1.1 | <i>A unifying reference framework for multi-target user interfaces</i> | 87 |
| 3.1.2 | <i>Multi-path UI development: UsiXML</i> | 89 |
| 3.1.3 | <i>Concrete User Interface</i> | 91 |
| 3.2 | AGENT AND MULTI-AGENT SYSTEMS | 96 |
| 3.2.1 | <i>Definition</i> | 96 |
| 3.2.2 | <i>Multi-agent systems design pattern</i> | 98 |
| 3.3 | ARCHITECTURAL DESCRIPTION | 100 |
| 3.3.1 | <i>General architecture</i> | 101 |
| 3.3.2 | <i>Shape recognition module</i> | 103 |
| 3.3.3 | <i>Shape interpretation module</i> | 111 |
| 3.4 | PRESENTATION OF THE APPLICATION | 118 |
| 3.4.1 | <i>Parameterize the application</i> | 118 |
| 3.4.2 | <i>Elements of the SketchiXML Environment</i> | 119 |
| 3.4.3 | <i>Interacting with SketchiXML</i> | 120 |
| 3.4.4 | <i>Building Widgets</i> | 121 |
| 3.4.5 | <i>Editing functions</i> | 122 |
| 3.4.6 | <i>Gesture training</i> | 123 |
| 3.4.7 | <i>Grammar edition</i> | 125 |
| 3.4.8 | <i>Level of fidelity</i> | 126 |
| 3.4.9 | <i>Navigation Editor</i> | 127 |
| 3.4.10 | <i>Preview</i> | 128 |
| 3.4.11 | <i>UsiXML output</i> | 129 |
| 3.5 | CONCLUSION | 131 |
| CHAPTER 4 A SUPPORT PROTOTYPE FRAMEWORK FOR DEVELOPMENT METHODOLOGIES | | 133 |
| 4.1 | REFERENCE FRAMEWORK | 133 |
| 4.1.1 | <i>High fidelity prototyping</i> | 134 |
| 4.1.2 | <i>Medium fidelity prototyping</i> | 137 |
| 4.1.3 | <i>Low fidelity prototyping</i> | 139 |
| 4.1.4 | <i>A prototyping framework</i> | 141 |
| 4.2 | APPLICATION TO DEVELOPMENT METHODOLOGIES | 142 |
| 4.2.1 | <i>The waterfall model</i> | 147 |
| 4.2.2 | <i>Spiral model</i> | 149 |
| 4.2.3 | <i>Extreme Programming (XP)</i> | 153 |
| 4.3 | CONCLUSION | 156 |
| CHAPTER 5 SURVEYS | | 159 |
| 5.1 | BUILDING A WIDGET CATALOGUE | 159 |
| 5.1.1 | <i>Participants</i> | 160 |
| 5.1.2 | <i>Methodology</i> | 160 |
| 5.1.3 | <i>Results</i> | 163 |
| 5.2 | TESTING THE APPLICATION | 165 |
| 5.2.1 | <i>Participants</i> | 165 |

| | | |
|---|---|------------|
| 5.2.2 | <i>Methodology</i> | 165 |
| 5.3 | EXPERIMENTAL STUDY ON FIDELITY LEVELS | 173 |
| 5.3.1 | <i>Participants</i> | 173 |
| 5.3.2 | <i>Apparatus and experimental task environment</i> | 174 |
| 5.3.3 | <i>Methodology</i> | 175 |
| 5.3.4 | <i>Results</i> | 175 |
| 5.3.5 | <i>Interpretation and discussion</i> | 177 |
| 5.4 | EVALUATING THE REPRESENTATIONS | 178 |
| 5.4.1 | <i>Widgets Taxonomy: an a priori classification</i> | 178 |
| 5.4.2 | <i>Current study objectives</i> | 180 |
| 5.4.3 | <i>Methodology</i> | 181 |
| 5.4.4 | <i>Results</i> | 182 |
| 5.5 | CASE STUDIES | 190 |
| 5.5.1 | <i>E-media</i> | 190 |
| 5.5.2 | <i>Find a movie</i> | 201 |
| 5.5.3 | <i>Designing a wizard</i> | 208 |
| 5.6 | USER TESTING LIMITATIONS | 213 |
| 5.7 | CONCLUSION | 214 |
| CHAPTER 6 CONCLUSION | | 217 |
| 6.1 | CONTEXT OF THIS WORK | 217 |
| 6.2 | CONTENT OF THIS DISSERTATION | 218 |
| 6.3 | VALIDATION | 219 |
| 6.3.1 | <i>External Validation</i> | 219 |
| 6.3.2 | <i>Internal Validation</i> | 220 |
| 6.4 | CONTRIBUTIONS | 227 |
| 6.5 | FUTURE WORK | 228 |
| 6.5.1 | <i>Extending the coverage of sketching artifacts</i> | 229 |
| 6.5.2 | <i>Improving the Text Divider</i> | 230 |
| 6.5.3 | <i>Tuning the Recognition Engine more extensively</i> | 230 |
| 6.5.4 | <i>Support for Multi-windowing Design</i> | 230 |
| 6.5.5 | <i>Augmenting the Support for Design Memory</i> | 231 |
| 6.5.6 | <i>Extending to other domains than Computer Science</i> | 231 |
| 6.5.7 | <i>Extension of UCWorkBench [Ucwo]: a requirements engineering tool</i> | 233 |
| 6.5.8 | <i>General Improvement of sketching facilities</i> | 235 |
| APPENDIX A - USIXML 1.8 CLASS DIAGRAMS | | 237 |
| APPENDIX B - USIXML COMPLIANT TOOLS | | 242 |
| APPENDIX C – WIDGETS CATALOGUE | | 251 |
| APPENDIX D – USIXML SPECIFICATION | | 257 |
| APPENDIX E – SKETCHIXML USER GUIDE | | 273 |
| REFERENCES | | 291 |

To my grand-fathers, Bon-papa Louis Lemoine and Dr. Papy Francis Coyette

Acknowledgement

I would like to express my thanks to:

- My advisors, Professor Jean Vanderdonckt and Professor Manuel Kolp, for their constant support.
- My colleagues from IAG school of management at Université catholique de Louvain.
- My family and Amélie for their permanent enthusiasm and support.
- Professors Pierre Leclercq, Laurence Nigay and Stéphane Faulkner for accepting to participate to the jury of this dissertation.
- Professor J.A. Jorge and his team for allowing us to use JavaSketchIt and the CALI library in our research
- Suzanne Kieffer and Mickaël Nicolay for conducting the surveys and providing the results.
- My friends.
- The WIST (Wallonie Information Société Technologies) program under convention n°031/5592 RW REQUEST.

Abstract

Designing the right User Interface (UI) of an Information System the first time is very unlikely to occur. Instead, UI design is recognized as a process that remains intrinsically *open*, *iterative*, and *incomplete*. Most designers consider hand sketches on paper as one of the most effective means to represent the first drafts of a future UI. This kind of unconstrained approach presents many advantages: sketches can be drawn at anytime, it is fast to learn and quick to produce, it lets the sketcher focus on basic structural issues instead of unimportant details, and it encourages creativity. The idea of developing a computer-based tool for sketching UIs naturally emerged from these observations. Such a tool would extend the advantages provided by sketching techniques by: easy creating, deleting, updating or moving of UI elements, thus encouraging typical activities in the design process such as exploratory design, checking and revision. In this thesis, we introduce SketchiXML, a multi-platform interactive application that enable designers and end users to sketch user interfaces with different levels of fidelity and support for different contexts of use. The results of the sketching are analyzed to produce interface specifications independently of any context. These specifications are exploited to progressively produce one or several interfaces, for several contexts of use. Moreover, this tool is integrated in a complete prototyping solution that can provide effective support to most software development methodologies.

Chapter 1 Introduction

Interactive applications are typically composed of two main parts: a functional core which contains the various semantic functions (or methods) of the application and the User Interface (UI) which gives access to these functions. If the first aspect attracted much interest in the past, very little attention was given to the problem raised by UI development as this part was treated as any other piece of software. This can be surprising, as a UI seems to be an indispensable component of any interactive software. It determines how easy a user input data, navigate among them, and control semantic functions of a software. Thus, a software equipped with powerful functions but a low quality UI may be under-exploited or misused. Several figures suggest that the UI part is important:

- The amount of Lines of Code (LOC) for a UI may represent from **50% to 70 %** of the total application code [Myer00].
- In a waterfall development life cycle, the time devoted to UI design, implementation, and evaluation respectively represents 45%, 50%, 37% of the total development time, which gives an average number of 44% of the total time devoted to the UI [Boeh88].
- In an interactive application, the UI is probably the portion which affects the most the general acceptability of the system by end users [Niel93].

Given the above importance of UI design, industrial and scientific communities have dedicated significant effort on the development of new techniques to reduce the time needed to obtain the right user interface.

Chapter 1 Introduction

As a result, many graphical editors, also called user interface builders, came on the scene for most of the existing programming and mark-up languages to develop the final UI faster. Although these tools have been proved to be very efficient for building a UI, designers were still looking for a precise methodology to guide them in the steps required to have a UI development life cycle resulting into a quality UI.

Many researchers working on that topic shared the same source of inspiration to rationalize the UI development method [Pala97, Unge96, Chat99, Pate00] primarily the UI design step: Software Engineering (SE). Undoubtedly, SE presents all the aspects required for a UI development method, as it is recognized to be structured, principle-based, and relying on explicit design knowledge.

Unfortunately, the attempt to bridge the Human-Computer Interaction (HCI) discipline and the software engineering domain have raised a set of hardly manageable difficulties, as the standard approaches for UI development was mainly empirical, experience-based, and relying on implicit knowledge. Many attempts for bridging the gap between HCI and SE have been conducted, but still nobody knows exactly how this should be achieved. Despite huge efforts, researches have never reached the expected results when trying to bridge both domains. [Limb05] summarizes some reasons why such a bridge did not produce the expected results:

- *Lack of rigor*: The development life cycle of interactive systems found in HCI does not necessarily involve the same level of rigor that is typically reached in SE [Brow97]. In addition, HCI development life cycle is estimated to involve an order of complexity higher than those found in SE since UI development does not adhere to an algorithmic approach [Wegn97]. Many attempts have been done to bring formal methods in HCI for this purpose [Pala97].
- *Lack of systematization*: as SE dreamed of a well-structured methodology for developing highly complex systems and to prove their correctness, so did HCI for developing UIs. However, the systematization, and the reproducibility found in SE methods cannot be transposed straightforwardly in HCI: the development life cycle remains inherently open, ill-defined, and highly iterative [Sumn97] as opposed to the domain of SE where it is structured, well-defined, and progressive [DSou99].

Chapter 1 Introduction

- *Lack of a principle-based approach:* SE development methods typically define system development as a series of stages or steps according to well-established principles such as rigor and formality, scalability, incrementality, separation on concerns.... In contrast, HCI usually progresses in a more opportunistic way when the current result is usable enough to proceed to the next stage [Puer97].
- *Lack of explicitness:* the knowledge required to properly conduct the UI development is not as principled as in SE, but also is it implicitly maintained in the mind of experienced designers. This knowledge is therefore harder to communicate from one person to another, although initiatives exist that make this knowledge more explicit through design patterns, usability guidelines, etc. [Szek96, Pate00]. Even more, when this knowledge is made more explicit, nothing can guarantee that it is applied uniformly and consistently within the same development project or across various development projects.

With respect to these issues, it turns out that the research for a new UI development method that addresses the aforementioned shortcomings was maybe attacking the problem from the wrong side. Rather than trying to define yet another method for UI development, it could be more realistic and appropriate to focus on existing techniques and provide an effective support for these methods which are already well established in corporate environments. Based on the existing techniques, attention should be paid on the best manner to get the right design rather than getting the design right.[Tohi06] Based on this observation, we try to understand what are the problems really faced during UI design and how designers manage them.

The first major observations came from Nanard and Nanard [Nana95] when they reported that the development life cycle of an interactive application consists of a sophisticated process that does not always proceed linearly in a predefined way. They present it as an interconnected set of development paths continuously alternating bottom-up and top-down approaches.

From this consideration, any development method (or methodology) or development tool is expected to effectively and efficiently support a flexible development life cycle, which does not lock the mental process of expert designers in a fixed procedural schema. Additionally, since UI design is likely to involve moderately experienced designers and end user (e.g., in participatory

Chapter 1 Introduction

design), the method and its supporting tool should enforce a minimum number of priority constraints.

[Sumn97] and [Luo95] emphasize the observation made by Nanard and Nanard as they consider the UI development process, as usually conducted in HCI, to be eminently **open**. On one hand, several development steps can be conducted or considered simultaneously, and on the other hand the process is **ill-structured** as the initial requirements are often **incomplete, ambiguous**, with **poorly defined goals** and vary depending on multiple variables, among which time is the most determinant. Unsurprisingly, this process is **iterative** because conducting any development step does not produce output which remains definitive. In contrast, new elements come into play that directly affect the output produced by this development step, thus requiring the output to be updated by conducting the development step again, or a sub-step of it. In practice, the most prevailing UI development method consists of a series of cycles being iterated. This process is referred to as **iterative design** [Cons99].

Another crucial and obvious aspect is **creativity**, which is often claimed by designers as a key aspect of their role that could not be automated or reproduced equally [Xiao02]. The authors suggest that the design phase should allow the designers to concentrate on their creative ideas instead of symbols used to deliver their thoughts. However, most of the current software used for this purpose require designers to input graphic components using mouse/keyboard with lots of toolbar buttons or menu items for selection. This approach based on a predefined set of objects and commands seems to be contradicting with creative task performance.

Moreover, the **User Centered Design (UCD)** paradigm [Cons99] suggests that new constraints are introduced in the UI development life cycle to support the participatory design. UCD aims at fostering the participation of users in designing and evaluating systems, in order to obtain products that suit better to users' expectations. UCD is important because:

- In a study of 74 interactive software development projects in industry and academia [Myer02], 87% of interviewed designers and developers reported a use of UI iterative design.

Chapter 1 Introduction

- Successful interactive software that is commercialized today have adopted UCD in their development life cycle, thus demonstrating that it significantly affects the final results.
- UCD engages the user, its activities and its environment in all stages of an interactive application development. Although user involvement in the UI development life cycle provides a very valuable input, the end user may be unable to understand all the method steps and techniques that will be used to ensure UCD.

With regards to these elements, most designers usually agree that **prototyping** of either the interactive application [Boar84] or its UI [Baum96] are activities to be considered seriously. Prototyping provides a potential answer to most shortcomings outlined earlier and enables, amongst all, discovering in a short time the gap between the user's requirements and the UI specifications or the UI itself [Snyd02, Plim04, Berk00].

Several types of prototyping tools exist today that achieve different goals. The rapid prototyping is often recommended to be the easiest way to interact with the end user. This technique is used to conduct the requirements elicitation from the end user perspective so as to structure, refine and present them in a convenient manner [Boar84]. Applied to the UI, it consists in having precise specifications of what is needed as fast as possible. To this aim, the designer prefers to avoid coding any UI, and waste time on expensive activity, and focusing on more affordable means. The primary goal of UI prototyping is therefore to reduce the cost and risk involved in developing the UI [Szek96, Mccu06]. The solution adopted should be preferably based on graphical representation, rather than on abstract specifications, so as to remove any potential barrier between the end user and the designer.

Among all prototyping techniques, hand sketches on paper turns out to be one of the most effective ways to represent the first drafts of a future UI [Bail03, Land01, Newm03, Snyd04, Lim06]. Sketching ideas on paper represents a familiar and unconstrained approach with many advantages: sketches can be drawn at any design stage, it is fast to learn and quick to produce, it is easy to modify, it lets the sketcher focus on basic structural issues instead of secondary details (e.g., exact alignment, typography, and colors), it is very appropriate to convey ongoing, unfinished designs, and it encourages creativity, sketches can be performed collaboratively between designers and end-users.

Furthermore, the end user may herself produce some sketches to initiate the development process and when the sketch is close enough to the expected UI, an agreement can be signed between the designer and the end user, thus facilitating contract and validation. Paper prototyping therefore appears to be a very viable answer to the requirements gathered earlier, since this approach does not impose any constraint on the representation and does facilitate the participation of end users as no complex or rigid semantics are considered.

Creating a low-fidelity UI prototype, such as UI sketches, is at least 10 to 20 times easier and faster than its equivalent with a high-fidelity prototype (such as produced in UI builders) [Vand02]. Indeed, low-fidelity UI prototype allows to dramatically decrease the time and resources needed for completing a development cycle, which is decomposed into three main stages: *design* when a new UI design comes into play, *prototype* when the new UI design is made concrete, and *evaluate* when the concrete prototype is evaluated against user' requirements (Figure 1-1).

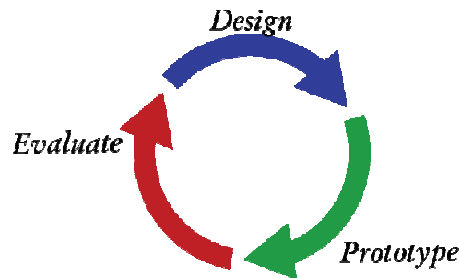


Figure 1-1 A simplistic representation of the spiral development cycle model [Vand02]

Unfortunately, paper prototyping is not the ultimate technique for UI prototyping as non-negligible drawbacks also exist. For instance, when sketching on paper, changes are hard to accomplish as the design evolves because of the intrinsic format of the paper support; the designer has to redraw the common features that the design retains... But the major drawback seems to be the lack of interaction between paper-based design and user [Land95]. In order to have a global overview of the interaction, a designer needs to "play the computer" (e.g., through a Wizard of Oz technique) and manipulate several sketches in response to a user's verbal or gesture actions [Rett94]. Another drawback lies in the way the end users consider paper sketch: the design process seems to be reduced to a set of small sketches and thus seems to be too simple and cheap to do anything valuable.

Chapter 1 Introduction

Given these observations, we consider that some improvement could be brought to the user interface design process. Rather than developing a brand new methodology we would like to present another approach based on the existing methodologies.

Consequently, the contents of this dissertation will be twofold:

1. The primary goal of this thesis consists of identifying the tools that are used for UI rapid prototyping. Indeed, prototypes are frequently used as they proved to be very helpful and valuable, but it appears that the tools support is far from being totally adequate. What are the current alternatives for a designer? To address the difficulty of modifying an existing paper prototype, prototyping software have introduced editing functions. The idea of developing a computer-based tool for sketching UIs naturally emerged from these observations [Hong01, Plim04, Szek96]. Such tools would extend the advantages provided by sketching techniques by easily creating, deleting, updating or moving UI elements, thus encouraging typical activities in the design process [Sumn97] such as checking and revision. Some research [Plim04, Land95, Caet02] was carried out in order to propose a hybrid approach, combining the best of the hand-sketching and computer assisted interface design. Despite these efforts, all the solutions provided do not appear as promising as wished: most of the tool tend to address one of the issue usually faced with current approaches but none of the tool propose a general solution that could be efficiently applied to existing methodologies. For instance, one tool may propose a very good compromise between paper prototype and computer assisted design in term of sketching, but once the sketch is completed someone has to code the UI in a programming language since the output is just a static picture. Other tools offer the possibility to instantly generate the corresponding code in Java or other language, but do no support the scenario editing [Plim04, Caet02]. Therefore, we will detail the drawbacks that were identified on existing prototyping tools and requirements to be met for a more suitable prototype based application will be listed. Based on this requirements list, with a tool for UI rapid prototyping, named SketchiXML, will be discussed
2. The second goal consists in proposing a way to integrate the UCD and prototyping in existing methodologies. Nowadays, UI development methodologies and software development methodologies are flourishing: iterative design, user centered design, iterative user centered design... the

purpose of this document is to integrate, in the latest methodologies, the systematic use of prototyping as a mean of requirements gathering and participatory design between the major stakeholders of the development. Rather than starting the development process from the top with a high level description of the application, we consider that the opposite approach is more relevant. Describing the application at a high level, involve the use of specific notations, tools, concepts hardly understandable for end users. Most of them do not know anything about the applications they are frequently using. How many end users wonder what programming language was used to develop the application they are using? I may be wrong, but I would not bet on a large percentage. So, questioning the end user on high level functionalities only is maybe inadequate since it requires a level of abstraction that is not familiar to everybody. Providing comments on an application in term of presentation is much more natural to most of the computer user. Everybody can make comments on a user interface and evaluate its usability, without the precise terms used by specialists or with solid argumentation.

1.1 Thesis

The following sub-sections describe the contents of this thesis. Firstly we provide a description of the thesis statement. Then, we explain how the validity of this thesis will be assessed. Lastly, the scope of this work is defined.

1.1.1 Thesis statement

This thesis demonstrates how designers could interact more efficiently with end users using low fidelity communication mean. The thesis proposes a new concrete approach, part of a complete prototyping framework, to support the current methodologies thanks to a better integration of the prototyping techniques.

This dissertation favors the use of low fidelity prototype as a communication means between the end users and the designers. Throughout this document, a better integration of the prototyping techniques and the current UI methodologies and software development will be recommended. To this end, the aforementioned shortcomings related to prototyping tool and methodologies will be discussed and two contributions will be achieved:

1. **A sketching tool for user interface prototyping.** SketchiXML aimed at solving all shortcomings identified in the existing tools into a single tool, allowing the designer to sketch the user interfaces as easily as on paper. In addition, the output generated is independent of any programming language as it generates UI specifications written in UsiXML (User Interface eXtensible Markup Language – <http://www.usixml.org>) [Limb05]), a platform-independent User Interface Description Language (UIDL) that will be exploited to produce code for one or several UIs, for one or many contexts of use simultaneously. Moreover, this tool is the only prototyping tool allowing smooth switching between several levels of fidelity.
2. **Add-in to existing methodologies:** the goal of this thesis is not to propose yet another new methodology; instead we propose to integrate the low fidelity prototyping based on SketchiXML in the existing methodologies. Thanks to the functionalities of the application, the time needed between the different iterations can be drastically reduced. Indeed moving from the low fidelity prototype to a runnable version is very fast.

1.1.2 Validation

Two kinds of validation are provided to assess the validity of this thesis:

First, the *external validation* will test the performance of the application in real situation on a large number of users of any types. To this end, several survey were conducted during this thesis in order to evaluate the performance of the application and to improve the drawbacks that were identified during this tests.

Second, the *internal validation* of the tool consists in assessing its characteristics against a set of selected criteria. The relevant criteria or requirement, for our tools are elicited after the state of the art of Chapter 2.

1.1.3 Scope

The scope of this thesis is delineated by the following hypotheses:

- The first hypothesis of this dissertation is to primarily cover the early design of graphical user interfaces. Indeed, UIs are an essential part of any application as they nearly always used in recent application. Moreover, we

Chapter 1 Introduction

aim at covering UI prototyping for multiple contexts of use; the *context of use* is defined as a triple of the form (E, P, U) where E is an envisioned or existing environment considered for a software system, P is any computing platform, and U is a user stereotype [Thev01]. This is consistent with the observation that the number of context of use is really exploding these days.

- The second hypothesis of this dissertation focuses on a specific kind of software i.e., information systems. An *information systems*(IS) is “a means of recording and communicating information to satisfy the requirements of all users, the business activities they are engaged in and the objectives established for them”. Such computer based system represent the majority of the entire set of systems, its proportion is estimated around 70% [Olle88]. IS deals with the practical and theoretical problems of providing information to an organisation and its members using computer systems, thus a video game cannot be considered as an IS.
- This dissertation predominantly targets the research community and to people involved in web design and early prototyping. Indeed, web designers are the ones that are likely to use prototype the most frequently.

1.2 Reading Map

In addition to the introduction and the conclusion, this dissertation is organized in five chapters.

Chapter 2 first we explore the different types of prototype and the different development paths for prototyping. Based on the literature on prototyping, we extend the theoretical framework with a set of new concepts and we propose a reference framework for prototyping. Then, we report on some significant work related to the paradigm of user centered design and early prototyping tools. Based on a presented evaluation grid, we survey in this chapter 14 different approaches and try to identify and compare their conceptual content along with their design process. For each tool, a set of advantages and shortcomings will be identified so as to be used to establish a list of requirements to be addressed. This requirements list will help us to assess the appropriateness of our solution.

Chapter 1 Introduction

Chapter 3 presents the technical aspects of SketchiXML. The multi agent architecture for shape recognition is presented, and the different libraries used for this purpose are detailed. This chapter also introduces the USIXML (USer Interface eXtensible Markup Language) language, a user interface specification language allowing describing user interfaces independently of any computing platforms or modalities. Many tools and other particularities of this language are also presented in this chapter. The last concept presented in this chapter is the agent paradigm. Indeed, SketchiXML is based on a multi-agent system providing support for the shape recognition and interpretation. For this purpose with introduce the paradigm and a set of design patterns used for multi-agent system development.

The gesture recognition algorithm that was designed especially for SketchiXML is also thoroughly explained, initially designed for signature recognition, this algorithm was enhanced to match SketchiXML specific requirements. The shape interpretation process is also presented with a special attention to the grammar edition module, allowing to specify custom representation for the widgets. This part of the application takes also advantage of the multi-agent technology, allowing to provide real time interpretation of complex shapes combination. Following the technical presentation of the tool, we provide an extensive presentation of SketchiXML based on a set of captures.

Chapter 4 shows how this tool could be integrated to existing methodologies. Based on the prototyping reference framework described in chapter 2, we propose a set of UsiXML tools supporting this approach. Based on this framework, we provide a set of recommendations to be applied when using this prototyping framework with traditional methodologies. This chapter is concluded with examples of well known methodologies extended with the prototyping framework.

Chapter 5 presents all the tests that were carried out during this work. The purposes of these tests were headed to validate the approach proposed thanks to usability tests and performances benchmarks. Additionally, a test phase was conducted to evaluate the performances of the user with regards to the type of representation used (fidelity level). In order to give a better understanding of SketchiXML and validates the general diverse case studies are illustrated for different context of use and different development path. Amongst all, the first case study illustrated consists in the e-commerce web site development that will be used in the state of the art presented in Chapter 2.

Chapter 1 Introduction

Chapter 6 concludes by discussing the appropriateness of the solution proposed in this dissertation and addressing the validity of this work. Our contributions are summarized and some ideas for future work and extensions are proposed.

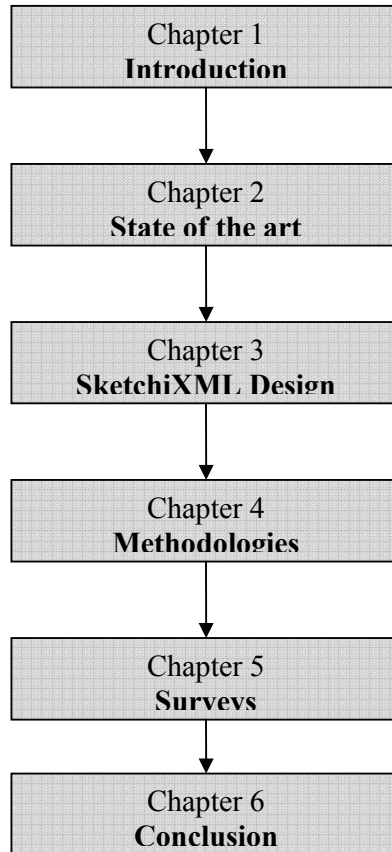


Figure 1-2. Thesis reading map

Chapter 2 State of the art

As stated in the introduction, several tools are already available for UI computer assisted design. The following section introduces the different alternatives existing in computer-aided sketching tool. All of the tools and techniques presented are based on the same case study: e-media, a web site selling digital media, books and more. This chapter is divided into six subsections.

Section 1 introduces the different kinds of prototypes; we detail the prototyping paths that can be considered for user interface prototyping and the level of fidelity.

The second subsection introduces the comparison grid defined for comparing tools providing support for UI prototyping. This grid contains a set of basic information such as the author name, required libraries...etc and all the relevant criteria that should be considered when comparing such applications.

In order to highlight the differences between computer assisted design tools and non-computer assisted design alternative, we explain first what we define as the classical approaches. These approaches mainly based on pens, pencils, paper, glue, scissors, post-it... are described through the second subsection.

The fourth subsections present the prototyping tools, each tool is described with a short description, a set of captures to present the tool itself and the output generated and eventually we conclude with a short summary of the main advantages and drawbacks of this tool. All the tools presented in this section address the UI prototyping process, thus high fidelity editor such as Borland JBuilder, Macromedia Dreamweaver or .net framework are not presented as they are only aimed at building concrete UI.

The fifth subsections briefly introduces a set of tools not directly linked with early prototyping of user interface, but addressing issues directly or indirectly linked to our domain of interest.

Finally, we conclude this chapter with a general summary of the previous subsections. We structure all the data gathered in the previous subsection into a single and shortest grid. Based on the previous analysis we elicit the requirements that should be met by the new prototyping tool.

2.1 Prototyping

The difference between prototyping and mock-ups in the literature does not appear straightforward. Therefore, the following definition is adopted.

A *prototype* is a model of the system delivered in the medium of the system. For example, a web site prototype would be delivered as a web site, using the standard web protocols, so that it could be interacted with in the same medium as the project's product. Prototypes do not have to be fully functioning; they merely have to be illustrative of what the product should look and feel like [Clem99].

In contrast, we consider a *mock-up* as a UI representation delivered in a medium that is different from the system. A web site mock-up might be a paper-based representation of how the pages should look like. Another web site mock-up could be a representation drawn collaboratively on a white board.

Apart from the medium, the difference between a mock-up and a prototype is difficult to characterize. Indeed prototyping can be divided into several levels of granularity, ranging from low-fidelity to high-fidelity. The definition of a low-fidelity prototype is interpreted similarly to the description of a mock-up. Therefore, the mock-up and low-fidelity prototype concepts will be used indifferently to define the early prototyping phase, and (late-) prototype to define a running prototype of the application.

The following subsections introduce a set of concepts related to the prototyping. The first subsection covers the level of fidelity associated to a prototype. Based on this description, prototyping in various development paths, which are not necessarily linear, could be analyzed.. The second subsection briefly defines the different kinds of prototype while the third subsection examines a standard

Chapter 2 State of the Art in Informal Design

approach for prototyping development. The two remaining subsections detail the scope of the prototype and the prototype executability: a concern of high importance.

2.1.1 Levels of fidelity

The *low-fidelity (Lo-Fi) prototype* is a limited function and limited interaction prototype. They are constructed to depict concepts, design alternatives, and screen layouts rather than to model the user interaction with the system. The most standard approaches for Lo-Fi prototyping are the “paper and pencil technique”, the “whiteboard/blackboard and post-it approach” [Vand02]. Such approaches provide access to all the components, and prevent the designer from being distracted from the primary task of design. This type of prototyping lets you iterate through an entire cycle of design, prototype, and evaluate in less than a day [Rett94].

The *medium-fidelity (Me-Fi) prototype* consists in building a UI mock-up giving importance to the content, but keeping secondary all the information regarding the typography, color scheme or others minor details. A typical example is Microsoft Visio [Micr07] where only the type, the size and the contents of UI widgets can be specified graphically. Medium fidelity prototypes are a good compromise when a mockup representation is required.

The *high-fidelity (Hi-Fi) prototype* is a set of screens that provide a dynamic, computerized, working model of the planned system. High fidelity prototyping tools are thus equipped with a wide range of editing functions for all UI widgets: erase, undo, move, specify physical attributes, etc... This kind of software allows the designer to build a complete GUI from which is produced an accurate image (e.g., Adobe Photoshop, PowerPoint) or the code in a determined programming language (e.g., Visual Basic, DreamWeaver).

A low level of fidelity is applicable during the early stage of the development life cycle of an interactive application, especially when the specifications of the user interface are still unknown, incomplete and still need to be discovered. Under these conditions, the purpose is mainly to explore the design alternatives without going into details. As this step is repeated frequently, it must stay light in term of time and money.

Oppositely, a highest fidelity level is more appropriate for the late stage of development. Either, the domain is sufficiently understood and mastered so as to

Chapter 2 State of the Art in Informal Design

propose a user interface sufficiently close to the expected final user interface; or the repeated iteration of the prototype have permitted to identify the relevant aspect to integrate in the final result. The purpose is then to refine the current prototype until we obtain the final user interface as finalized as possible. Consequently the cost and production time is higher.

| | Low | Medium | High |
|----------------------------|--|--|---|
| Content | Mainly presentation | Presentation, content, basic navigation | Presentation, navigation, content, layout, functionalities |
| Use, discovery | Exploration, evocation, communication, discovery | Simulation, refinement, iteration, improvement, usability validation, user testing | Final specifications, documentation, marketing, propagation to the application |
| Type of prototype | Horizontal | Diagonal | Vertical |
| Cost | Low | Average | High |
| Time | Low | Average | High |
| Approach | Bottom-up | Middle-out | Top-down |
| Naturalness | Very high | Average | Low |
| Detail level | Low | Average | high |
| Iteration frequency | Very high | Average | Low |
| Appearance | Sketchy Little visual detail | - Simple - medium level of detail, close to the appearance of the Final UI | -Definitive, refined - Look and Feel of final UI |
| Advantages | - Low development cost - Short production time - Easy communication - Basic drawing skills needed | - Medium development cost - Average production time - May involve some basic graphical aspects as specified in style guide: labels, icons,... - Limited drawing skills - Understandable for end user | - Fully interactive - Serves for usability testing - Supports user-centered design - Serves for prototype validation and contract - Attractive for end users - Code generation |
| Shortcomings | - Is facilitator-driven - Limited for usability tests -Limited support of navigational aspects - Low attractiveness for end users - No code generation | - Is Facilitator-driven - Limited for usability tests - Medium support of navigational aspects - No code generation | - High development cost - High production time - Advanced drawing and specification skills needed - Very inflexible with respect to changing requirements |

Table 2-1 Summary of the different levels of fidelity

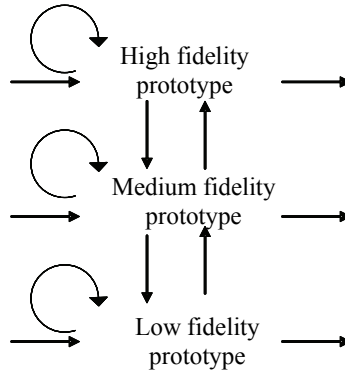


Figure 2-1 Prototype development paths

The prototype at a high level of fidelity may be consecutive to a lower fidelity prototype but not necessary (Fig. 2-1). All the development paths for prototyping are possible in theory [Vand06] with any initial point and any ending point. In principle the prototype can be initiated from any level of fidelity as long as it corresponds to the end user needs and it can end at any level. Practically, we mostly observe development paths starting from low fidelity to a high fidelity in order to support an iterative and progressive prototype. The iteration can occur at any level, but as the level of fidelity increase simultaneously with the time needed to build the prototype, the number of iteration should decrease. Also, there is no need to go through all the levels, depending of the type of user interface to be built some paths will be more appropriate.

2.1.2 Prototyping development paths

The prototypes differ according to their levels of realism. A horizontal prototype only presents the visible part of the software: the windows or the home page for a web site. It allows conducting a perception test. Then, the main functionalities of the application are developed in a vertical prototype in order to run user tests. Based on these two notions, a new development path for prototype that is diagonal is introduced: it alternates between horizontal and vertical prototyping.

An interactive application can be divided into three layers: the UI, the abstraction and control layer and the functional kernel. This kind of architecture can be found in the typical MVC (model – view – controller) [Beck87] or PAC (presentation – abstraction – controller) patterns [Cout87].

Nielsen distinguishes two levels of prototyping according to the level of interaction provided by the prototype [Mack93]. These two levels are presented hereunder, and are extended with other alternative for prototyping path:

a. Horizontal

The horizontal prototype is the 'surface interface'. Software functionalities are not working, but it allows achieving a perception test. It can even be a sketch on paper. It is a prototype that models many features but with little details. It can be seen as a horizontal slice of a system structure chart from the top down to a specific depth.

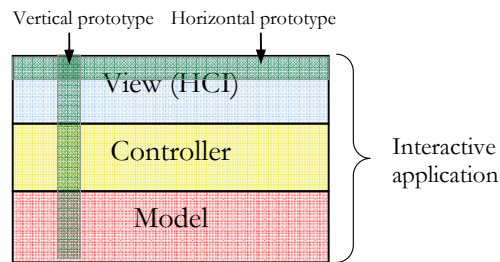


Figure 2-2 Illustration of the horizontal and vertical prototyping process

Such prototype allows conducting the perception test consisting in evaluating the ease of understanding the interface. The windows are shown to the user. Without letting him using the mouse, the observer is asking the user to explain how he understands the information displayed and which sort of behavior is expected. This first stage allows to check the local behavior of the interface and to identify the critical points where usability problems are likely to appear. Such technique is thus very helpful in the first stages of the design, since it allows to quickly collect relevant information. The purpose is to test the overall interaction metaphor, so includes common functions that the user is expected to perform frequently. We will try to prototype as many functionalities of the application through its user interface.

b. Vertical

A vertical prototype implements a consistent set of functionalities in order to allow the user to achieve a typical use of scenario. Some of these functionalities could be simulated. It can be seen as a vertical

slice of a system structure chart from top to bottom. Obviously, such approach appears to be more useful in the later design stages.

This second stage of the prototyping phase consists in a series of user tests details of the design during which the critical points previously raised are evaluated.

User testing allows to identify usability problems and to analyze their cause with the users. Solutions are elaborated and implemented in the following version of the prototype which will be the subject of a new series of tests and so on.

c. Diagonal

Unfortunately, things are not always that clear. When developing a project, attention must be given to the past experience of the designer. Indeed the designer might be used to build some functionalities (functionalities already developed in other projects previously) while other are completely unknown (impossible to reuse past development, no system like this exist, ...). Under these specific conditions, it seems natural to consider other alternatives combining both approaches in what we would call a “diagonal prototype” [Vand06]. In this kind of prototype all the functionalities that are already well mastered are developed with vertical prototype while the functionalities that are not mastered are submitted to a horizontal prototyping phase. By doing so, the advantages of both approaches can be coupled without to suffer from the drawbacks. In this situation, the propagation of the prototype is called “expensive” as the prototype can spread in all directions. That is the reason why the medium fidelity prototype seems more appropriate for diagonal prototyping while a low level of fidelity is most suitable for a bottom-up approach and a high fidelity prototype is more adequate with a top-down approach.

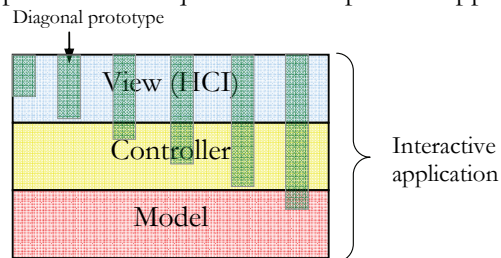


Figure 2-3 Illustration of the diagonal prototyping process

d. Specific to user interface

We can pursue the description with a decomposition of the human-computer interface layer in three parts: the presentation, the local navigation and the global navigation. Local navigation refers to the internal navigation of a UI, i.e. actions performed on components that change the content of the UI. Global navigation refers to the navigation from one UI to another. Based on this new splitting, new paths can be considered. The most frequent situation consists in initiating the prototyping of the user interface with the easiest part, the most visual and the more natural for the end user: the presentation of the information. This kind of prototype is called a presentation prototype (2.1.3.d). Figure 2-4 shows a larger cover of the presentation phase that reflects the important role given to the presentation aspect of the user interface in this prototype.

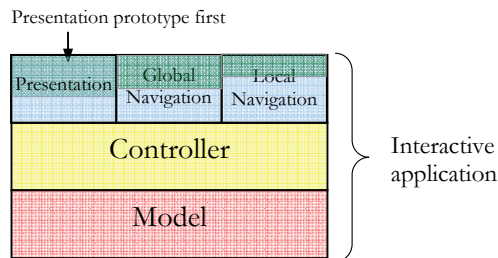


Figure 2-4 Presentation prototype first

Less frequently, the prototype process is initiated from the global navigation. In such perspective, the designer elaborates an architecture of interaction units or information specified with their goals and linked together with regards to the informational needs of the end users. As the global navigation evolves, attention can be dedicated to the presentation of the interaction unit and some elements of the local navigation can be defined. Figure 2-5 illustrates such process; it appears that most of the efforts are concentrated on the global navigation, then the presentation and eventually the local navigation.

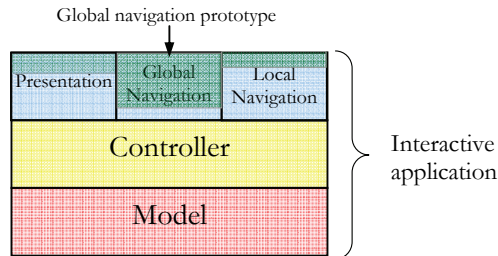


Figure 2-5. Global navigation prototype first

Finally, a third alternative consist in starting the prototype with the local navigation aspect. In this case, the end user specifies the interaction he desires with the particular interaction units, i.e. specify the order of a sequence of dialog in a wizard. Once the local navigation is defined, this information is imported in the presentation prototype and eventually the process closes on the definition of the major elements of the global navigation. This development path is presented on Figure 2-6

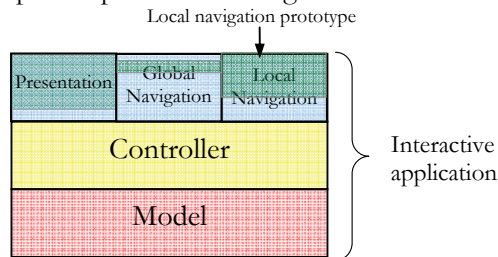


Figure 2-6 Local navigation prototype first

2.1.3 Prototyping types

Another aspect of interest of the prototyping relies in its type. Indeed, software prototyping appears to have many variants even if some are considered more frequently than other. Among the 6 types of prototyping approaches presented in this subsection, the first three approaches corresponds to the Floyd classification known as the 3E model that is widely accepted and used [Floy84]. This model classifies the three main approaches generally considered for prototyping.

a. Throwaway Prototyping (Exploratory)

Throwaway or Rapid Prototyping is the most easily understood prototyping method. After preliminary requirements gathering is

accomplished, a simple working model of the system is constructed to visually show the users what their requirements may look like when they are implemented into a finished system. Such a prototype can be used to clarify project goals, to examine alternative designs, or to investigate a large and complex system. It results in discussions of what should be achieved by a task and how it can be supported with the current techniques.

Rapid Prototyping involves creating a working model of various parts of the system at a very early stage, after a relatively short investigation. The method used in building it is usually quite informal, the most important factor being the speed with which the model is provided. The model then becomes the starting point from which users can re-examine their expectations and clarify their requirements. When this has been achieved, the prototype model is 'thrown away', and the system is formally developed based on the identified requirements.

Throwaway prototyping are not necessary low fidelity prototype even if the paper prototyping seems to be the most natural approach for building throwaway prototype. A method to easily build high fidelity throwaway prototypes is to use a GUI Builder and create a click dummy, that means a prototype that looks like the target system, but does not provide any functionality.

Developing such prototype consists naturally in developing a horizontal prototype as no functional aspect needs to be developed at this stage.

b. Evolutionary Prototyping

Evolutionary Prototyping is quite different from Throwaway Prototyping. The main goal when using Evolutionary Prototyping is to build a very robust prototype in a structured manner and constantly refine it. "The reason for this is that the Evolutionary prototype, when built, forms the heart of the new system, and the improvements and further requirements will be built on to it" [Crin91].

When developing a system using Evolutionary Prototyping, the system is continually refined and rebuilt. "...evolutionary prototyping acknowledges that we do not understand all the requirements and builds only those that are well understood"[Davi92]. This technique allows the development team to add features, or make changes that could not be conceived during the requirements and design phase. But in order to capture the first requirements, the use of low fidelity makes sense. These two approaches appear thus to be complementary since they do not focus on the same issues.

Such prototype can be considered as a diagonal prototyping, all the aspects of the applications will be developed incrementally. User interface and functionalities development will alternate along the process.

c. Functional prototypes (Experimental)

Functional prototypes implement strategically important parts of both the user interface and the functionality of a planned application. Contrary to the presentation prototype, this kind of prototype is mainly vertical, even if some user interface development need to be done in order to address the issue of interest. So, functional prototype could be considered either as diagonal prototype or vertical prototype depending on the importance of the user interface in the issue addressed.

d. Presentation prototypes

Presentation prototypes are built to illustrate how an application may solve given requirements. As they are often used as part of the project proposal, they are strongly focused on the user interface. Naturally, this kind of prototype is always a horizontal prototype.

e. Breadboards

Breadboards serve to investigate technical aspects such as system architecture or system functionality of a planned application. They

are built to investigate certain aspect of special risk. They are not intended to be evaluated by end users. Similarly to the functional prototype, this kind of prototype is mainly vertical.

f. Pilot systems

Pilot systems are very mature prototype which can be practically applied.

In addition to this classification, prototype can also be classified according to other criteria such as the scope associated to the prototype. Even if this aspect is implicitly present in the previous definitions, a clear definition is proposed in next subsection.

2.1.4 Scope of prototyping

a. Local

A prototype of a single usability-critical system component

- a vertical prototype that is focused on one feature
- useful at some specific stage of the design process

b. Global

prototype of the entire system

- an expanded horizontal prototype that models a greater number of features and covers multiple levels of the system's structure chart
- useful throughout the design process

2.1.5 Prototype executability

Will the prototype be runnable and, if so, what does that mean? Indeed several types of interaction techniques between the prototype and the users exist. Here are the five alternatives usually considered.

a. Chauffeured prototype

This kind of prototype "runnable" in the very loose sense that the prototype allows a walkthrough to be performed. Typically, the

designer walks through with the user and manually demonstrates how the interface would respond to user actions. For example, the user might say "I'd click this button", and the designer would pull out a dialog box on paper that would appear. The advantage of a chauffeured prototype is that not all pieces need to be assembled but interactivity can still be tested; the designer can spontaneously create any missing pieces based on what the user needs in any given scenario.

b. Animation prototype

This kind of technique is "runnable" in the loose sense that it is executed frame by frame in "slide show" mode on a computer. For instance a designer designing a web site using Microsoft PowerPoint, obtain a set of slides for each user interface and the possible transition between the user interfaces. Then, Microsoft PowerPoint permit to emulate the navigation between the screens, even if the user interfaces are just empty shells.

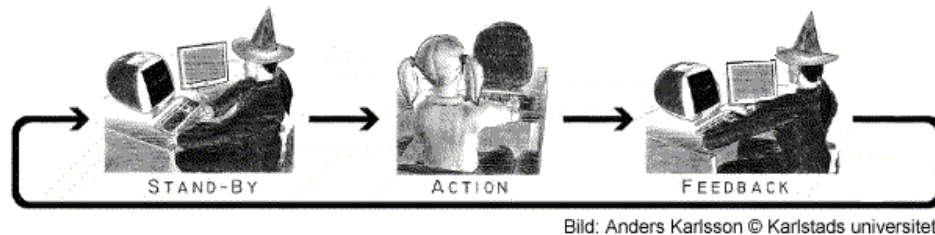


Figure 2-7 Illustration of the wizard of oz experiment, the user test the application when a designer (wizard on the picture) emulates the interaction of the application

c. Wizard of Oz

A wizard of Oz prototype is "runnable" in the sense that it executes in "slide show" mode but allows a third party, hidden from view that is pulling the levers and flipping the switches. The wizard of oz technique in user testing has a user interacting with an interface without knowing that the responses are being generated by a human, not a computer. This allows testing of some difficult interface concepts before a system is fully working.

For example, in a search system, a user may type in a query, and an expert behind the scenes rewrites the query in a formal syntax or hand-selects search results. This allows you to test theories in query formulation and filtering of results. Similarly, the wizard of oz technique is popular in testing natural language interfaces where, for instance, the choice of what syntax to support in the system is driven by what syntax users actually use during the tests.

d. Interactive prototype

This is a runnable prototype in the strict sense that it executes on the computer and responds to user input in real time but do not perform any computations. It corresponds to the horizontal prototype, only the visible layer is developed.

e. Functional prototype

This is a runnable in the very strict sense that it executes on the computer, responds to live input, and performs some of the expected computations.

Throughout this section we have presented most of the prototyping alternatives; depending of the goal pursued, the prototyping choice appears thus of high importance. These concepts will be useful for the understanding of this thesis, as the prototyping aspects are very redundant.

2.2 Analysis grid

The analysis grid is divided into six main categories:

1. *The tool identification section* contains all the relevant information on the tool such as its name, url, manufacturer...
2. *Install and first use section* contains all the information related to the installation of the tools. What are the hardware and software requirements to run the application correctly...

Chapter 2 State of the Art in Informal Design

3. *The general functions section* details the general functionalities such as the export functionalities, the code generation, the possibility to describe the navigational aspects...
4. *Shape recognition and shape interpretation section* are related to the previous category. If the previous category identifies shapes recognition and interpretation, then these categories provide information on the process to be applied for this purpose.
5. *The UI editor section* describes the UI editor in terms of functionalities, the editing function, handwriting capabilities, layout mechanism... are listed in this last category.

| | | | |
|---------------------------|-----------------------|----------------------------------|--|
| pe inte rpr ated | Tool Identification | Tool Name | Tool name |
| | | Tested Version | Version x.x |
| | | Last Version | Version x.x |
| | | Company Name | Company name |
| | | Brief description | Brief description |
| | | URL | www.sketchtool.org |
| | | Licensing | Open source or not / type of license |
| | Install and First Use | Required libraries | List of required libraries |
| | | Required software | List of required software |
| | | Recommended hardware | List of recommended input/output devices, minimum processing power |
| | | Install time | None = that the software is provided with an auto-install procedure |
| | Tool functions | Shape recognition | Yes or no |
| | | Shape interpretation | Interpret shape with respect to a specific context |
| | | Code or specification generators | VB, java, XHTML, C# |
| | | Level of fidelity | High, average, low |
| | | Navigation editor | Enables to specify navigation walkthrough corresponding to different scenarios |
| | | Preview | Is it possible to switch to a run mode or a simulation |
| | | Pattern manager | Stores template description |
| | | Usability adviser | Criticize the usability while sketching the UI |
| | Shape recognition | Shape recognition library | Cali, Rubine's algorithm... |
| | | Type of shapes | Rectangle, diamond, ellipse, circle, line, arrow, cross,... |
| | | Flexibility of recognition | Multiple stroke shapes and single stroke shapes. Closed forms or open forms. Dashed lines. Rotated shapes. |
| | | Performance | We propose two metrics: accuracy and average recognition time |
| | Conceptual coverage | | Concrete presentation (widgets), navigation links |

Chapter 2 State of the Art in Informal Design

| | | |
|-----------|---|--|
| | Number of recognized elements (widgets) | Absolute number of recognized widgets |
| | Interpretation mechanism | Grammar, Bayesian grammar, not known |
| | Process type | Batch mode= shapes are interpreted in sets after a certain time, run-time mode = shapes are interpreted directly |
| | Disambiguation mechanism | Contextualized (takes into account surrounding elements), non-contextualized. A contextualized disambiguation enables the possibility of having elements composed of multiple shapes |
| | Extensibility of interpretation mechanism | Some tools allow the extension of the exiting interpretation mechanism |
| | User adaptive grammar | Choice of a grammar depending on user preferences |
| UI Editor | Interaction style | Pen-based, property sheets |
| | Layout | Absolute coordinates, relative coordinates, embedded boxes mechanism |
| | Granularity | Multi-window/frame, Mono-window/frame, . |
| | Zooming | Possibility to zoom |
| | Hand writing recognition | Does the tool support handwriting recognition |
| | Editing functions | Does the tool support a wide set of editing functions such as copy, paste, move |
| | Colors | User defined or fixed |
| | Assistance | Contextualized help, manual,... |
| | Language | Mono or Multi-language |

Table 2-2 Evaluation grid for early design tools

2.3 Classical approaches

As the two approach depicted in the following section are very similar, the evaluation grid is only applied to the first alternative as the results are almost the same for the two approaches considered hereunder. Indeed, the main difference between these two approaches lies in the re-use of pre-constructed component. In the same way, we do not detail the white board approach as it is very similar to the two methods presented in the following subsection.

2.3.1 The Paper prototyping

[Snyd02] presents paper prototyping as a useful method of usability testing for Web sites, Web applications, and conventional software. The main principles are the following:

Chapter 2 State of the Art in Informal Design

- You first decide on the tasks that you'd like the user to accomplish.
- Next, you make screen shots and/or hand-sketched drafts of the windows, menus, dialog boxes, pages, popup messages, etc. that are needed to perform those tasks.
- Then you conduct a usability test by having one or two developers play the role of "computer," manipulating the pieces of paper to simulate how the interface would behave. Users are given realistic tasks to perform by interacting directly with the prototype -- they "click" by touching the prototype buttons or links and "type" by writing their data in the prototype's edit fields. (Using transparency or removable tape prevents the prototype from being written on directly.)

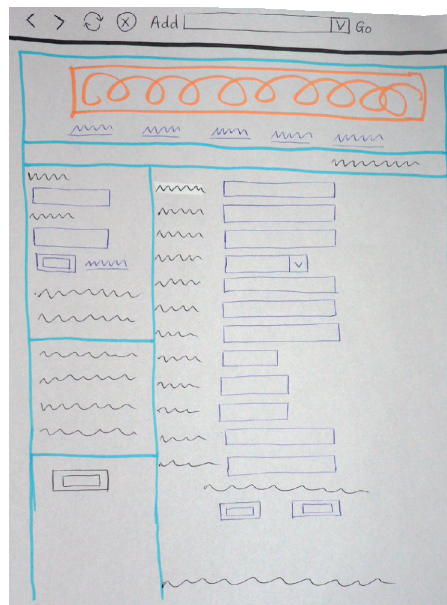


Figure 2-8 An example of paper prototype

A facilitator (usually someone trained in usability) conducts the session while other members of the development team observe and take notes. The “computer” does not explain how the interface is supposed to work, but merely simulates what the interface would do. In this manner, you can identify which parts of the interface are self-explanatory and which parts are confusing.

Chapter 2 State of the Art in Informal Design

In addition to the relative complexity associated to the run-mode, paper prototyping also shows a set of drawbacks such as the difficulty to accomplish changes or the need to redraw the common features that the design retains.

a. Advantages:

- ↗ Support for scenario-based design
- ↗ Inexpensive
- ↗ Few constraints on widget representation
- ↗ Very natural
- ↗ No preparation required
- ↗ No prerequisite knowledge needed

b. Shortcomings:

- ↘ No shape recognition and interpretation, thus losing the effort
- ↘ No code generation
- ↘ No preview mode
- ↘ “Run-mode” need several persons
- ↘ After several iteration on the same sheet of paper a deterioration of the support might occur
- ↘ Sketching not scalable, difficult to have a global overview of UIs
- ↘ Changes hard to accomplish

c. Tool Specifications

| | | |
|-----------------------|----------------------|---|
| Tool Identification | Tool Name | Paper prototyping |
| | Tested Version | - |
| | Last Version | - |
| | Company Name | - |
| | Brief description | Prototype the future interfaces on paper |
| | URL | http://www.snyderconsulting.net |
| | Licensing | Free |
| Install and First Use | Required libraries | Not applicable |
| | Required software | Not applicable |
| | Recommended hardware | Not applicable |
| | Install time | Not applicable |
| 1 function | Shape recognition | no |

Chapter 2 State of the Art in Informal Design

| | | |
|-----------|----------------------------------|---|
| | Shape interpretation | no |
| | Code or specification generators | no |
| | Level of fidelity | no |
| | Navigation editor | Yes |
| | Preview | Mimics a run mode |
| | Pattern manager | no |
| | Usability adviser | no |
| UI Editor | Interaction style | Pen |
| | Layout | Absolute coordinates, relative coordinates (embedded boxes mechanism) |
| | Granularity | Multi-window/frame, Mono-window/frame, . |
| | Zooming | Not applicable |
| | Hand writing recognition | Not applicable |
| | Editing functions | Yes |
| | Colors | Gum |
| | Assistance | Free |
| | Language | Not applicable |

Table 2-3 Evaluation grid for paper prototyping

2.3.2 Tiny fingers prototyping

The tiny fingers method presented in [Rett94] use of simple materials and equipment in order to create a paper-based simulation of an interface or system. Paper prototypes provide a valuable and cost-effective means to evaluate and iterate design options before a team gets committed to one implementation. Interface elements such as menus, windows, dialogues and icons can be sketched on paper or created in advance using card, acetate, pens etc.

When the paper prototype has been prepared a member of the design team sits before a user and ‘plays the computer’ by moving interface elements around in response to the user’s actions. The user makes selections and activates interface elements by using their finger as a mouse and writing ‘typed’ input. A further person facilitates the session by providing task instructions and encouraging the user to express their thoughts and impressions. [Rett94]

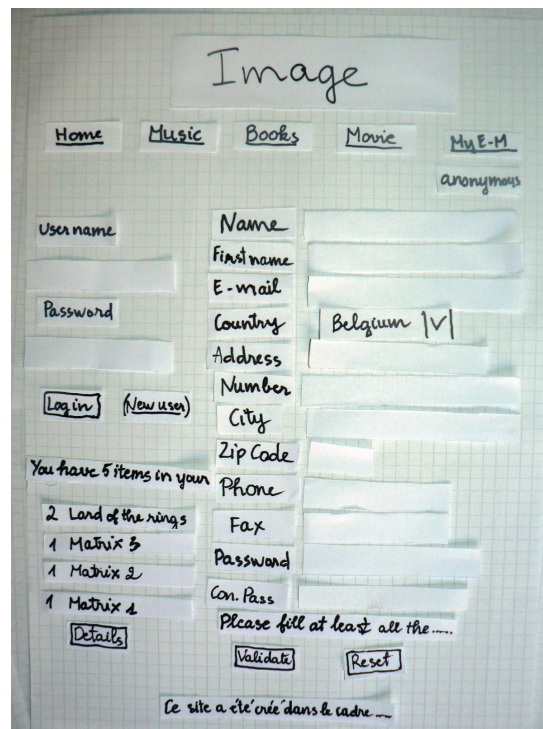


Figure 2-9 An example of UI build with the tiny fingers method

a. Advantages:

- Support for scenario-based design
- Inexpensive
- Few constraints on widget representation
- Very natural

b. Shortcomings:

- No shape recognition and interpretation, thus losing the effort
- No code generation
- No preview mode
- Need to build all the widget in a first phase, even if a kit is available on the web.
- Difficulty to change the size of the pre-constructed widgets, explosion of the number of widget required.
- Difficulty to have a global overview for large set of UI

- ✧ Might become messy as number of paper increase very fast
- ✧ Not scalable
- ✧ “Run-mode” need several persons

2.4 UI Sketching Applications

Throughout this chapter we introduce a set of computer-based tools for sketching UIs naturally. As explained in the introduction, the need for this kind of application emerged from the observations elicited in Chapter 1.1. Such tools would add on top of the advantages provided by sketching techniques a wide range of advantages: easily creating, deleting, updating or moving UI elements, thus encouraging checking and revision, typical activities in the design process.

2.4.1 Silk

Silk (Figure 3-5) presented in [Land96] is a UNIX application based on a gesture library and a gesture recognition, based on the Rubine’s Algorithm, system that allows the designer to draw predefined widgets (typically, GUI widgets) and apply command by gesture on these widgets. Silk is also, able to interpret shapes in order to obtain a widget representation. It does so partially or totally. That is: if the designer wants to preserve the initial sketching, it is possible to cancel the interpretation and leaves the drawing as it is.

If the designer wants to recognize the shape, it can be transformed into its corresponding widget. Silk is also equipped with storyboarding capabilities: by drawing arrows, the designer can express mini-scenarios like: “if the end user clicks on this button, she will go to that window”, “if the end user selects this radio button, it will affect that push button”. Once recognized, Silk automatically generates a UNIX resource file containing the definition of the UI. If some objects or shapes have not been recognized, they are left out.

As this tool is not available anymore, Figure 2-10 shows an example that was proposed by the creator. Oppositely, the following tools are all based on the same case study.

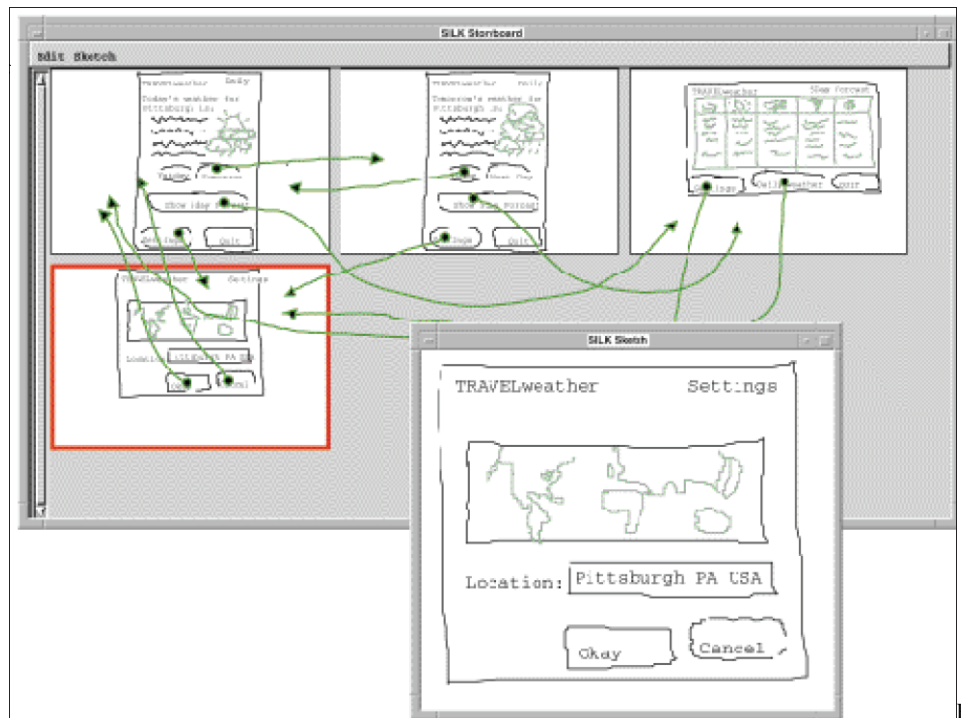


Figure 2-10 An example of UIs build Silk

a. Advantages:

- Support for scenario-based design
- Several levels of granularity
- Good documentation
- Gesture Library for sketch editing
- Mature product, based on experimental testing
- Zooming facility from local design (e.g. a web page) to a global design (e.g., a portion of a web site or n entire web site)
- Storyboarding facilities based on patterns
- Widgets Recognition
- Presentation close to the final user interface, but not runnable

b. Shortcomings:

- No code generation
- No preview mode
- Only dedicated to web sites

Chapter 2 State of the Art in Informal Design

c. Grid

| | | |
|-----------------------|---|---|
| Tool Identification | Tool Name | Silk |
| | Tested Version | 1.0 |
| | Last Version | Version 1.0 |
| | Company Name | Carnegie Mellon University |
| | Contact Person | James Landay |
| | Brief description | - |
| | URL | http://www.cs.berkeley.edu/~landay/research/publications/SILK_CHI/jal1bdy.html |
| | Licensing | No |
| Install and First Use | Required libraries | None |
| | Required software | Unix, OSF/Motif |
| | Recommended hardware | Pointing device |
| | Install time | None |
| Tool functions | Shape recognition | Yes |
| | Shape interpretation | Yes |
| | Code or specification generators | Yes |
| | Level of fidelity | low |
| | Navigation editor | Yes |
| | Preview | Yes |
| | Pattern manager | No |
| | Usability adviser | No |
| Shape recognition | Shape recognition library | Grammar |
| | Type of shapes | Circles, squares, rectangles. |
| | Flexibility of recognition | Low, due to Rubine's Algorithm limitations |
| | Performance | No estimated |
| Shape interpretation | Conceptual coverage | Widgets |
| | Number of recognized elements (widgets) | 10 |
| | Interpretation mechanism | Grammar |
| | Process type | Unknown |
| | Disambiguation mechanism | Not contextualized |
| | Extensibility of interpretation mechanism | Possibility to extend gesture library |
| | User adaptive grammar | Yes |
| UI Editor | Interaction style | Pen-based |
| | Layout | - |
| | Granularity | Multi-window/frame, . |

Chapter 2 State of the Art in Informal Design

| | |
|--------------------------|--------------|
| Zooming | Yes |
| Hand writing recognition | No |
| Editing functions | Yes |
| Colors | Yes |
| Assistance | User defined |
| Language | No |

Table 2-4 Silk Specifications

2.4.2 Denim

Denim [Lin2000, Land01] is the successor of Silk in its main principles, except for stroke recognition based on gestures. It is thus an informal pen-based system that helps web site designers in the early stages of design. It allows designers to quickly sketch web pages, view them at different levels of detail, create links among them, and interact with them in a run mode.

Denim supports sketching input with very little sketch recognition and allows design at different refinement levels, and unifies the levels through zooming.

Note that the last version of Denim (v 2.0 Beta ver8) tool proposes, in addition to sketching, a toolbox containing a set of generic widget representations as it can be seen on picture 2-11.

a. Advantages:

- ↗ Support for scenario-based design
- ↗ Several levels of granularity
- ↗ Good documentation
- ↗ Toolbox of generic widgets
- ↗ Mature product, based on experimental testing
- ↗ Zooming facility from local design (e.g. a web page) to a global design (e.g., a portion of a web site or n entire web site)
- ↗ Storyboarding facilities based on patterns

b. Shortcomings:

- ↘ No shape recognition and interpretation, thus losing the effort
- ↘ No code generation
- ↘ No preview mode
- ↘ Only dedicated to web sites

Chapter 2 State of the Art in Informal Design

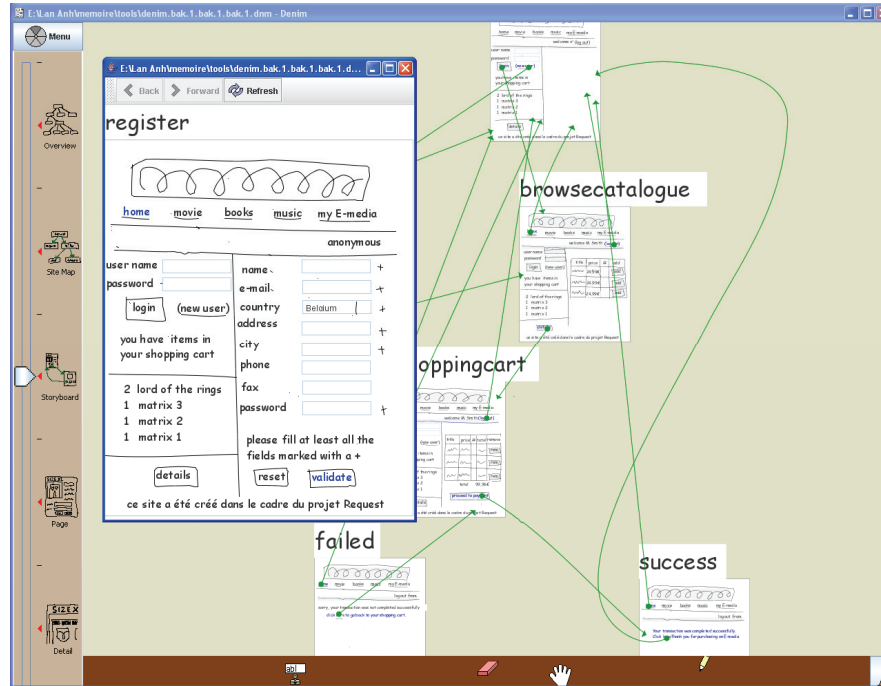


Figure 2-11 An example of UI build with Denim

c. Tool Specifications

| | | |
|-----------------------|----------------------------------|---|
| Tool Identification | Tool Name | Denim |
| | Tested Version | 1.1 |
| | Last Version | Version 2.0 Beta ver 8 |
| | Company Name | University of Berkley |
| | Contact Person | James Lin |
| | Brief description | - |
| | URL | http://guir.cs.berkeley.edu/projects/denim/ |
| | Licensing | No |
| Install and First Use | Required libraries | None |
| | Required software | Java 1.4.1_02 |
| | Recommended hardware | Pointing device |
| | Install time | None |
| Tool functions | Shape recognition | little |
| | Shape interpretation | little |
| | Code or specification generators | No |
| | Level of fidelity | No |

Chapter 2 State of the Art in Informal Design

| | | |
|-----------|--------------------------|--|
| | Navigation editor | Yes |
| | Preview | Yes |
| | Pattern manager | Yes, possibility to store user defined shapes and re-use them across projects. |
| | Usability adviser | No |
| UI Editor | Interaction style | Pen-based |
| | Layout | Free |
| | Granularity | Multi-window/frame, . |
| | Zooming | Yes |
| | Hand writing recognition | No |
| | Editing functions | Well defined editing functions |
| | Colors | User defined |
| | Assistance | Yes (documentation, help files) |
| | Language | English |

Table 2-5 Denim Specifications

2.4.3 Gabbeh

Gabbeh [Nagh04, Nagh05] is a prototype tool that extends the capabilities of existing tools by supporting dialogues between different designers, or between designers and other stakeholders. Gabbeh is an extension to Denim that allows different stakeholders to add arbitrary annotations in the form of comments either when the model is being designed or when the model is being executed. Comments can be associated with any arbitrary number of design components, such as panels, labels, texts and scribbles. Moreover, comments are given a background color to allow development teams to distinguish between different types of comments, or perhaps between comments from different speakers.

Gabbeh allows end-users to view and add comments while they are reviewing the design in ‘run mode’. This functionality is intended to allow end-users to give feedback through the prototyping medium.

a. Advantages:

- ↗ Support for scenario-based design
- ↗ Support for annotations
- ↗ Several levels of granularity
- ↗ Good documentation

Chapter 2 State of the Art in Informal Design

- Toolbox of generic widgets
- Mature product, based on experimental testing
- Zooming facility from local design (e.g. a web page) to a global design
- Storyboarding facilities based on patterns

b. Shortcomings:

- No shape recognition and interpretation, thus losing the effort
- No code generation
- No preview mode
- Only dedicated to web sites

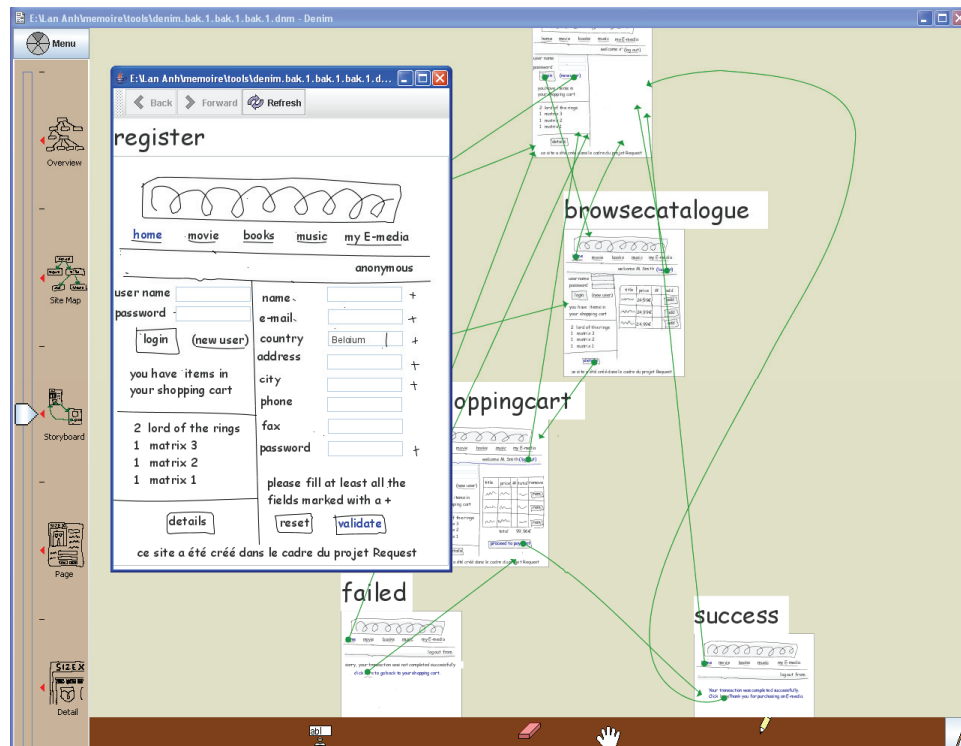


Figure 2-12 An example of UI build with Gabbeh

c. Tool Specifications

| | | |
|---------------------|----------------|--------|
| Tool Identification | Tool Name | Gabbeh |
| | Tested Version | 1 |
| | Last Version | 1 |

Chapter 2 State of the Art in Informal Design

| | | |
|-----------------------|----------------------------------|---|
| | Company Name | Sheffield Hallam University |
| | Contact Person | Amir M Naghsh |
| | URL | http://extra.shu.ac.uk/paperchaste/gabbeh-hci04.htm |
| | Licensing | No |
| Install and First Use | Required libraries | None |
| | Required software | Java 1.4.1_02 |
| | Recommended hardware | Pointing device |
| | Install time | None |
| Tool functions | Shape recognition | little |
| | Shape interpretation | little |
| | Code or specification generators | No |
| | Level of fidelity | No |
| | Navigation editor | Yes |
| | Preview | Yes |
| | Pattern manager | Yes, possibility to store user defined shapes and re-use them across projects. |
| | Usability adviser | No |
| UI Editor | Interaction style | Pen-based |
| | Layout | Free |
| | Granularity | Multi-window/frame, . |
| | Zooming | Yes |
| | Hand writing recognition | No |
| | Editing functions | Well defined editing functions |
| | Colors | User defined |
| | Assistance | Yes (documentation, help files) |
| | Language | English |

Table 2-6 Gabbeh Specifications

2.4.4 CrossWeaver

CrossWeaver [Sinh03] is a tool aimed at helping designers to prototype multimodal and multi-device user interfaces. This tool relies on the same paradigm than denim and prone the use of informal prototypes and to create a working prototype from these sketches. This prototypes can run across multiple standalone devices simultaneously, processing multimodal input from each one. CrossWeaver captures all of the user interaction when running a test of a prototype. This input log can quickly be viewed for the details of the users'

Chapter 2 State of the Art in Informal Design

multimodal interaction, and it can be replayed across all participating devices, giving the designer information to help him or her iterate the interface design.

For each individual UI, the designer can specify navigation based on multiple modalities, also, the designer can specify region of interaction and apply specific behavior to multimodal action done on these particular region.

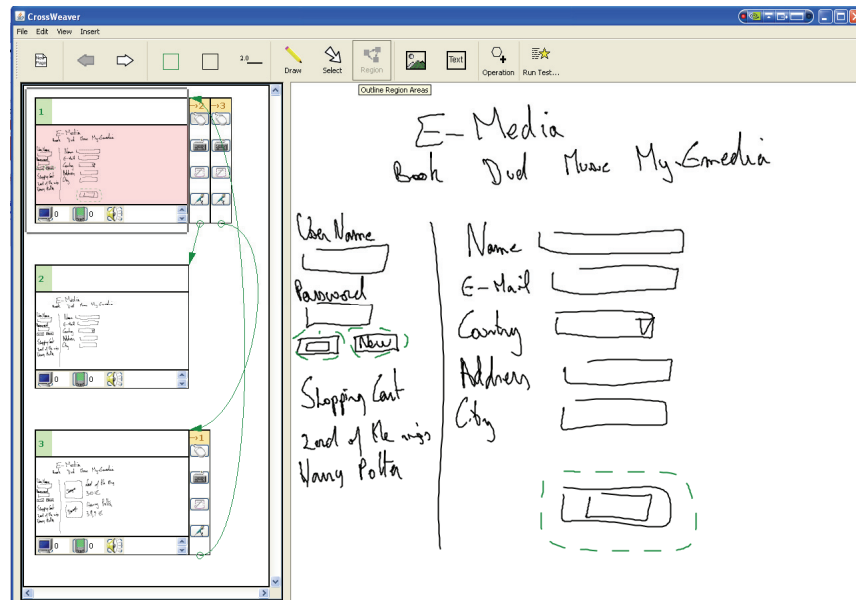


Figure 2-12 An example of UI build with CrossWeaver

a. Advantages:

- Support for scenario-based design
- Mature product, based on experimental testing
- Run mode
- Storyboarding allowing to specify multi-modal navigation

b. Shortcomings:

- No shape recognition and interpretation, thus losing the effort
- No code generation
- Only dedicated to web sites
- Editing functions could be improved, erasing should be easier
- No zooming facilities

c. Tool Specifications

| | | |
|-----------------------|----------------------------------|---|
| Tool Identification | Tool Name | CrossWeaver |
| | Tested Version | 1 |
| | Last Version | 1 |
| | Company Name | University of Berkley |
| | Contact Person | Anoop Sinha, James Landay |
| | URL | http://guir.berkeley.edu/projects/crossweaver/ |
| | Licensing | No |
| Install and First Use | Required libraries | None |
| | Required software | Java 1.4.1_02 |
| | Recommended hardware | Pointing device |
| | Install time | None |
| Tool functions | Shape recognition | No |
| | Shape interpretation | No |
| | Code or specification generators | No |
| | Level of fidelity | low |
| | Navigation editor | Yes |
| | Preview | Yes |
| | Pattern manager | No |
| | Usability adviser | No |
| UI Editor | Interaction style | Pen-based |
| | Layout | Free |
| | Granularity | Single window |
| | Zooming | No |
| | Hand writing recognition | No |
| | Editing functions | No zoom, delete uneasy |
| | Colors | User defined |
| | Assistance | Yes (documentation, help files) |
| | Language | English |

Table 2-7 CrossWeaver Specifications

2.4.5 JavaSketchIt

JavaSketchIt [Caet02] is a tool allowing UI prototyping by sketching the UI in a pen-based interaction style. JavaSketchIT is a visual approach to layout static components of UIs as hand-drawn compositions of simple geometric shapes,

Chapter 2 State of the Art in Informal Design

based on sketch recognition. The sketch recognition process is done thanks to the Cali library[Fons02]. This library is able to identify shapes of different sizes, rotated at arbitrary angles, drawn with dashed, continuous strokes or overlapping lines, and use fuzzy logic to associate degrees of certainty to recognized shapes to overcome uncertainty and imprecision in shape sketches

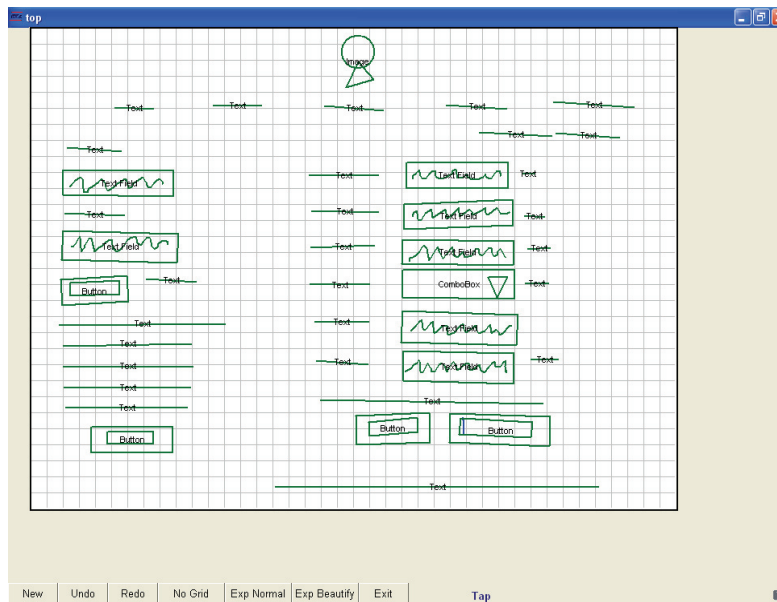


Figure 2-13 An example of UI build with JavaSketchIt

[Caet02] have defined a visual grammar using drawing data from target users, where they tried to figure out how people sketch interfaces and what combinations of shapes are more commonly used to define widgets. From these they built a grammar and implemented a prototype, JavaSketchIt, which allows creating UI through hand-drawn geometric shapes, identified by a gesture recognizer.

This prototype generates a Java interface, whose layout can be beautified using an a posteriori set of grammar rules (e.g. to align and group objects). Unfortunately, the layout used for the java UI is based on absolute coordinates.

a. Advantages:

- Performance (speed and accuracy)
- Multi-stroke gestures
- Recognizes rotated shapes

Chapter 2 State of the Art in Informal Design

- Computationally light
- Open source
- Requires standard and freely available libraries
- Extensible shape interpretation grammar

b. Shortcomings:

- ✗ Mono-window
- ✗ No scenario editor
- ✗ Only generates java (and no UI spec)
- ✗ Limited widget set
- ✗ Shape interpretation can only take as input a construct made of a maximum of two vectorial shapes
- ✗ No zoom

c. Tool Specification

| | | |
|-----------------------|----------------------------------|---|
| Tool Identification | Tool Name | JavaSketchIt 1.0 |
| | Tested Version | 1.0 |
| | Last Version | 1.0 |
| | Company Name | Instituto Technico Lisboa / INESC |
| | Contact Person | Joachim A. Jorge |
| | Brief description | - |
| | URL | http://immi.inesc.pt/project_page.php?project_id=21 |
| | Licensing | Yes, GNU licence |
| Install and First Use | Required libraries | Based on the Cali library (provided with the tool) |
| | Required software | JavaSketchIT, Java Run Time Environment. |
| | Recommended hardware | Tablet PC, pointing device. minimum PII 300Mhz |
| | Install time | None |
| Tool functions | Shape recognition | Yes |
| | Shape interpretation | Yes |
| | Code or specification generators | Java AWT + beautification (widgets are laid out elegantly at code generation) |
| | Level of fidelity | No |
| | Navigation editor | No |
| | Preview | Yes |
| | Pattern manager | No |
| | Usability adviser | No |

Chapter 2 State of the Art in Informal Design

| | | |
|----------------------|---|--|
| Shape recognition | Shape recognition library | Fuzzy logic (Cali libraries) |
| | Type of shapes | Rectangle, triangle, diamond, ellipse, circle, line, arrow, cross, V, wavy line (low oscillation movement), delete (high oscillation movement) |
| | Flexibility of recognition | Closed forms, open forms, dashed lines , rotated shapes |
| | Performance | Accuracy: 92% Average recognition time: less than 50 ms (using A Pentium II @ 233 MHz) |
| Shape interpretation | Conceptual coverage | Widgets |
| | Number of recognized elements (widgets) | 10 |
| | Interpretation mechanism | Explicit grammar |
| | Process type | Real time |
| | Disambiguation mechanism | Contextualized |
| | Extensibility of interpretation mechanism | Yes |
| | User adaptive grammar | No |
| UI Editor | Interaction style | Pen-based |
| | Layout | Absolute coordinates |
| | Granularity | Mono-window/frame, . |
| | Zooming | No |
| | Hand writing recognition | No |
| | Editing functions | Well defined |
| | Colors | Fixed |
| | Assistance | None |
| | Language | Mono (English) |

Table 2-7 JavaSketchIt Specifications

2.4.6 FreeForm 2

FreeForm2 [Plim02, Plim04] provides a pen-based interactive environment for drawing UI forms and then interacting with the design while it is rendered as a sketch. Freeform runs as a Visual Basic 6 add-in and can interpret sketched shapes and convert these shapes into a VB forms. FreeForm uses a single stroke recognizer and then a rule base and dictionary for combining simple strokes into

Chapter 2 State of the Art in Informal Design

Visual Basic widgets and words. The rule base also includes beautification size constraints. In addition to size, widgets are aligned on to a grid and grouped appropriately. Even if Freeform is integrated to Visual Basic consider it as a single tool and do not take all the possibilities of Visual Basic into account.

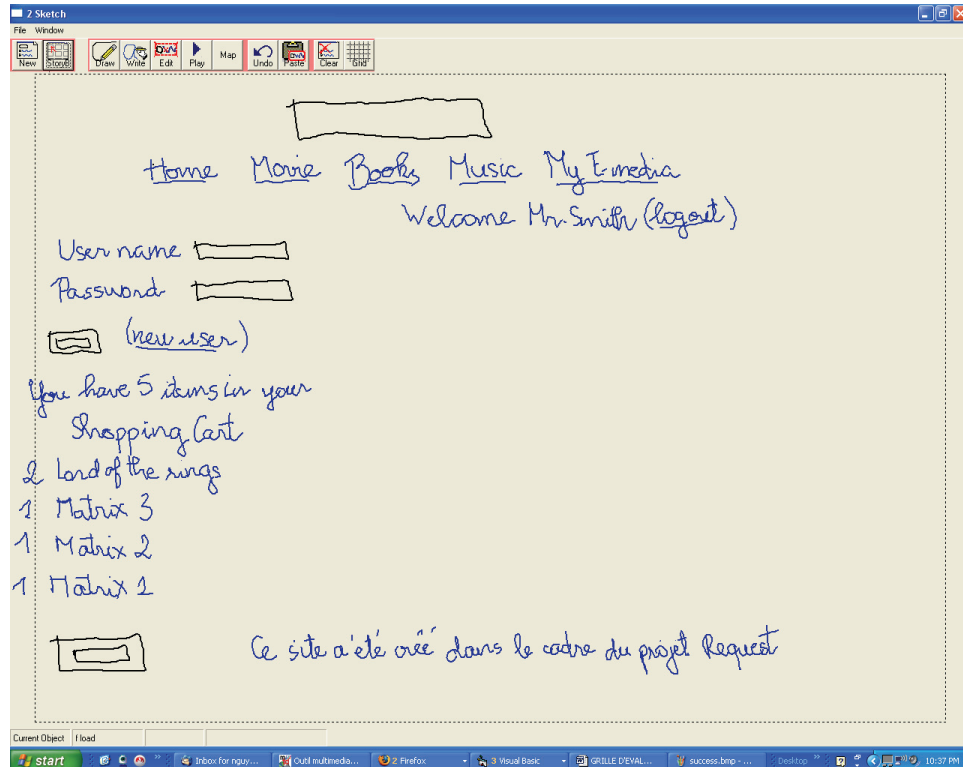


Figure 2-14 An example of UI build with free form

a. Advantages:

- Performance (speed and accuracy)
- Scenario editor
- Multi-windows
- Handwriting recognition (but not very precise)
- Possibility to specify navigation

b. Shortcomings:

- Only works with VB6
- Mono-stroke shapes

Chapter 2 State of the Art in Informal Design

- ✧ No rotated shapes
- ✧ Limited widget set
- ✧ Not open source
- ✧ Shape interpretation can only take as input a construct made of a maximum of two vectorial shapes
- ✧ No zooming possibilities

c. Tool Specification

| | | |
|-----------------------|---|---|
| Tool Identification | Tool Name | FreeForm 2 |
| | Tested Version | 2.0 |
| | Last Version | 2.0 |
| | Company Name | University of Oakland |
| | Contact Person | Beryl Plimmer |
| | URL | http://www.cs.auckland.ac.nz/~beryl/ |
| | Licensing | No |
| Install and First Use | Required libraries | Free Form libraries |
| | Required software | Visual Basic 6 |
| | Recommended hardware | Tablet PC, pointing device |
| | Install time | None |
| Tool functions | Shape recognition | Yes |
| | Shape interpretation | Yes |
| | Code or specification generators | Visual Basic 6 |
| | Level of fidelity | No |
| | Navigation editor | Yes |
| | Preview | Yes |
| | Pattern manager | No |
| | Usability adviser | No |
| Shape recognition | Shape recognition library | Rubine's algorithm |
| | Type of shapes | Not listed exhaustively. Rectangle, triangle, diamond, ellipse, circle, line, arrow, cross, V, wavy line (low oscillation movement), delete (high oscillation movement) |
| | Flexibility of recognition | Single stroke shapes, no support for rotated shapes |
| | Performance | Accuracy: 90% Average recognition time: not estimated |
| Shape interpretation | Conceptual coverage | Widgets + Navigation |
| | Number of recognized elements (widgets) | 10 with a maximum of two strokes/widget |
| | Interpretation mechanism | Implicit grammar |

Chapter 2 State of the Art in Informal Design

| | | |
|-----------|---|---|
| | Process type | Batch |
| | Disambiguation mechanism | Contextualized (only for grouping radio buttons together) |
| | Extensibility of interpretation mechanism | No information |
| | User adaptive grammar | No |
| UI Editor | Interaction style | Pen-based |
| | Layout | Absolute coordinates |
| | Granularity | Multi-window/frame |
| | Zooming | No |
| | Hand writing recognition | Yes (but limited, it recognizes lowercase characters formed with a single stroke) |
| | Editing functions | Well defined |
| | Colors | Fixed |
| | Assistance | None |
| | Language | Mono (English) |

Table 2-8 FreeForm Specifications

2.4.7 Inkkit

InkKit [Plim07], the successor of Freeform, is a sketch toolkit designed to support diagramming across a wide range of domains. It consists of two main components: UI and a customizable recognition engine. The UI has two main views: sketch pages and portfolios. On a sketch page the user can draw and write much as they would on a piece of paper, yet supported by usual computer editing functionality. The portfolio is a place where a collection of sketches is displayed and links can be used to establish relationships between sketches.

The main interest of InkKit is its recognition engine. Advanced recognition techniques mean that users can draw and write on a page without having to change modes. Recognition of a particular type of diagram is achieved by creating a diagram domain and providing a few hand-drawn examples of the different types of diagrams components. Software add-ins can be written to convert the recognized sketch to another format or support intelligent interaction with the sketch.

Unfortunately, we had some difficulties to test this application as the pre-beta version we received was still bugged: we did not manage to install this easily, and many editing function did not act as they were supposed to. However, based on what we have read and seen so far, the next versions should be promising.

Chapter 2 State of the Art in Informal Design

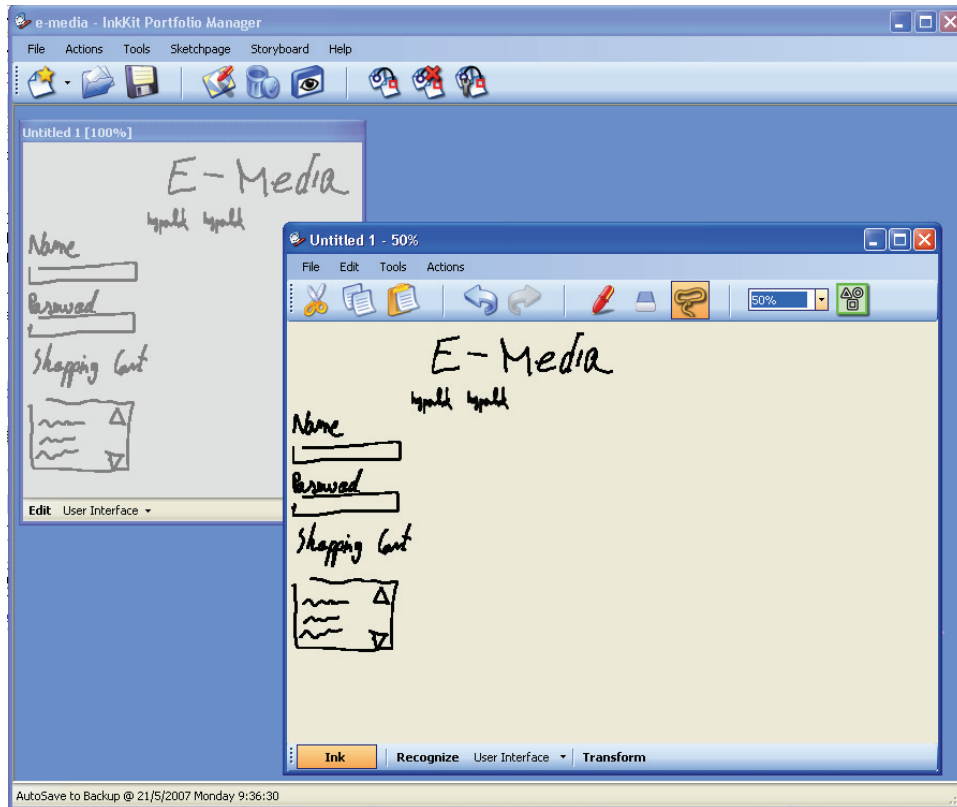


Figure 2-15 An example of UI build with InkKit

Even if based on the Rubine's algorithm, InkKit is able to recognize multi-stroke gestures. We interpreting the sketch, InkKit tries to group the strokes together in order to build more complex shape. Obviously, a lot of combinations have to be tested and it takes a lot of time. Real time recognition is thus incompatible with the approach adopted.

a. Advantages:

- Performance (speed and accuracy)
- Scenario editor
- Multi-windows
- Handwriting recognition
- Multi-strokes gesture recognizer
- Extendable

Chapter 2 State of the Art in Informal Design

b. Shortcomings:

- ✧ No rotated shapes (use of Rubine's algorithm)
- ✧ Extendable (require manual coding)
- ✧ Current version bugged, makes it difficult to install and use
- ✧ Recognition slow (many shapes combinations are tested)

c. Tool Specification

| | | |
|-----------------------|---|---|
| Tool Identification | Tool Name | InkKit |
| | Tested Version | 1 |
| | Last Version | 1 |
| | Company Name | University of Oakland |
| | Contact Person | Beryl Plimmer |
| | URL | http://www.cs.auckland.ac.nz/~beryl/ |
| | Licensing | No |
| Install and First Use | Required libraries | Microsoft Handwriting recognition pack 1.7 |
| | Required software | Microsoft .net |
| | Recommended hardware | Tablet PC, pointing device |
| | Install time | Some libraries to install, installation was straightforward |
| Tool functions | Shape recognition | Yes |
| | Shape interpretation | Yes |
| | Code or specification generators | Java, HTML... |
| | Level of fidelity | Low |
| | Navigation editor | Yes |
| | Preview | (rendering) |
| | Pattern manager | No |
| | Usability adviser | No |
| Shape recognition | Shape recognition library | Rubine's algorithm |
| | Type of shapes | User defined |
| | Flexibility of recognition | Multiple-strokes shapes, no support for rotated shapes |
| | Performance | not estimated |
| Shape interpretation | Conceptual coverage | Widgets + Navigation |
| | Number of recognized elements (widgets) | User defined (require manual coding for extension) |
| | Interpretation mechanism | Implicit grammar |

Chapter 2 State of the Art in Informal Design

| | | |
|-----------|---|---|
| | Process type | Batch |
| | Disambiguation mechanism | Contextualized |
| | Extensibility of interpretation mechanism | No information |
| | User adaptive grammar | No |
| UI Editor | Interaction style | Pen-based |
| | Layout | Absolute coordinates |
| | Granularity | Multi-window/frame |
| | Zooming | No |
| | Hand writing recognition | Yes (based on Microsoft recognition pack) |
| | Editing functions | Well defined |
| | Colors | Fixed |
| | Assistance | None |
| | Language | Mono (English) |

Table 2-9 InkKit Specifications

2.4.8 GUI Design Studio

GUI Design Studio [Car] is a graphical UI design tool for Microsoft Windows that you can use to rapidly create demonstration prototypes without any coding or scripting.

It permits to draw individual screens, windows and components using standard elements, connect them together to storyboard operational workflow then run the simulator to test your designs.

This commercial product is very mature as many version of the application were released in the past. The result provided consist in a medium fidelity output, many aspect of the UI can be represented, while only a small subset of attributes can be defined to each components.

a. Advantages:

- Automatic detection of the type of the prototype being built
- Easy integration with other office tools such as Microsoft Excel, data can be copied and pasted from one tool to the other very easily.
- Preview mode very convenient, conditions can be provided in order to test different alternatives.
- Easy to edit the properties of each of the elements present on the user interface. Interaction done via properties sheet.

Chapter 2 State of the Art in Informal Design

- Possibility to align elements easily

b. Shortcomings:

- No code generation
- Few widgets for web page creation
- Expensive

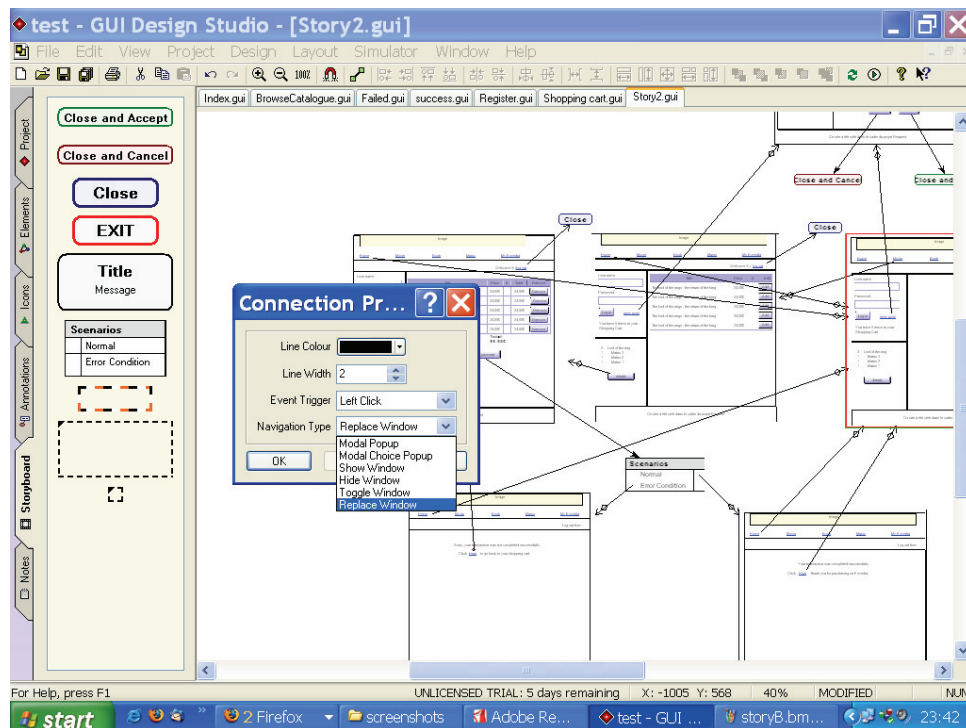


Figure 2-16 An example of UI build with GUI Design Studio

c. Tool Specification

| | | |
|---------------------|----------------|--|
| Tool Identification | Tool Name | GUI Design Studio |
| | Tested Version | 2.2.59.0 |
| | Last Version | 2.2.59.0 |
| | Company Name | Caretta Software Ltd |
| | Contact Person | Caretta Software Ltd 74 Meadow View Road Kennington Oxford OX1 5QX United Kingdom |
| | URL | http://www.carettaoftware.com/ |

Chapter 2 State of the Art in Informal Design

| | | |
|-----------------------|----------------------------------|------------------------------|
| Install and First Use | Licensing | \$497 |
| | Required libraries | none |
| | Required software | Microsoft Windows XP/NT/2000 |
| | Recommended hardware | Small configuration |
| | Install time | fast |
| Tool functions | Shape recognition | none |
| | Shape interpretation | none |
| | Code or specification generators | none |
| | Level of fidelity | High |
| | Navigation editor | Yes |
| | Preview | Yes |
| | Pattern manager | no |
| | Usability adviser | No |
| UI Editor | Interaction style | Mouse |
| | Layout | Absolute coordinates |
| | Granularity | Zooming functions |
| | Zooming | yes |
| | Hand writing recognition | None |
| | Editing functions | Complete coverage |
| | Colors | User defined |
| | Assistance | None |
| | Language | English |

Table 2-10 GUI Design Studio Specifications

2.4.9 Visio

Microsoft Visio [Micr07] is diagramming software for Microsoft Windows. It uses vector graphics to create diagrams and cover a wide set of different diagrams. As an example, Visio permits to build UML diagrams, but also to build medium fidelity mock up for user interfaces. In order to build the UI, the designer drag and drop the components needed and specify the attributes of each of the components.

Microsoft Visio, alike other Microsoft products, is well developed and very user friendly. Its usage is very similar to the other Microsoft tools and makes it easy to use. In addition to the basic functionalities that are understood by most of the end users, the designer has also the opportunity to use advanced features in order to simulate navigation for instance. This can be done thanks to the use of VBA language supported by Microsoft Visio.

Chapter 2 State of the Art in Informal Design

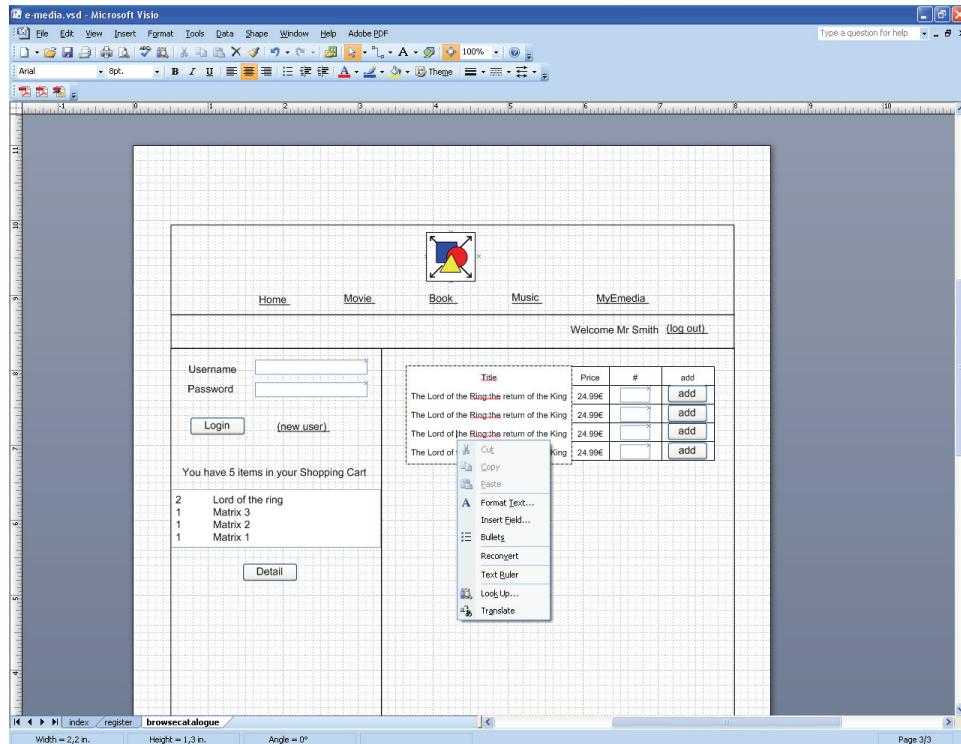


Figure 2-17 An example of UI build with Microsoft Visio

a. Advantages:

- Very mature product
- User friendly
- Good documentation
- Possibility to build several UIs simultaneously
- Possibility to add dynamic behavior with macro
- A new editor for a specific domain can be developed easily
- Easy to develop new plug-in

b. Shortcomings:

- ✗ No preview of the current work
- ✗ Navigation cannot be easily specified
- ✗ Expensive

c. Tool Specification

| | | |
|-----------------------|----------------------------------|---|
| Tool Identification | Tool Name | Microsoft Visio 2007 |
| | Tested Version | Visio 2007 |
| | Last Version | Visio 2007 |
| | Company Name | Microsoft Corporation |
| | Contact Person | Microsoft Corporation One Microsoft Way Redmond, WA 98052-6399 USA |
| | URL | http://office.microsoft.com/en-us/visio/HA101656401033.aspx |
| | Licensing | |
| Install and First Use | Required libraries | none |
| | Required software | Microsoft Windows |
| | Recommended hardware | 500 megahertz (MHz) processor or higher and 256 megabyte (MB) RAM or higher |
| | Install time | Low |
| Tool functions | Shape recognition | none |
| | Shape interpretation | none |
| | Code or specification generators | none |
| | Level of fidelity | medium |
| | Navigation editor | none |
| | Preview | none |
| | Pattern manager | Can be added |
| | Usability adviser | no |
| UI Editor | Interaction style | Mouse / keyboard |
| | Layout | Absolute coordinates |
| | Granularity | multi windows |
| | Zooming | Well supported |
| | Hand writing recognition | Not applicable (keyboard) |
| | Editing functions | Well defined |
| | Colors | User defined |
| | Assistance | Forums, online help... |
| | Language | Many languages proposed |

Table 2-11 Microsoft Visio Specifications

2.4.10stpBA Storyboarding

stpBA Storyboarding [StpBA] is a Microsoft Visio based requirements tool to capture and visually validate requirements with users through GUI storyboarding. The tool can be used standalone or integrated with stpsoft Quew, IBM Rational RequisitePro or Borland CaliberRM. In addition to the functionality of Microsoft Visio present before, such as easy UI prototyping, stpBA Storyboarding permits to build GUI storyboards so as to help end users to visualize how an application will behave.

Chapter 2 State of the Art in Informal Design

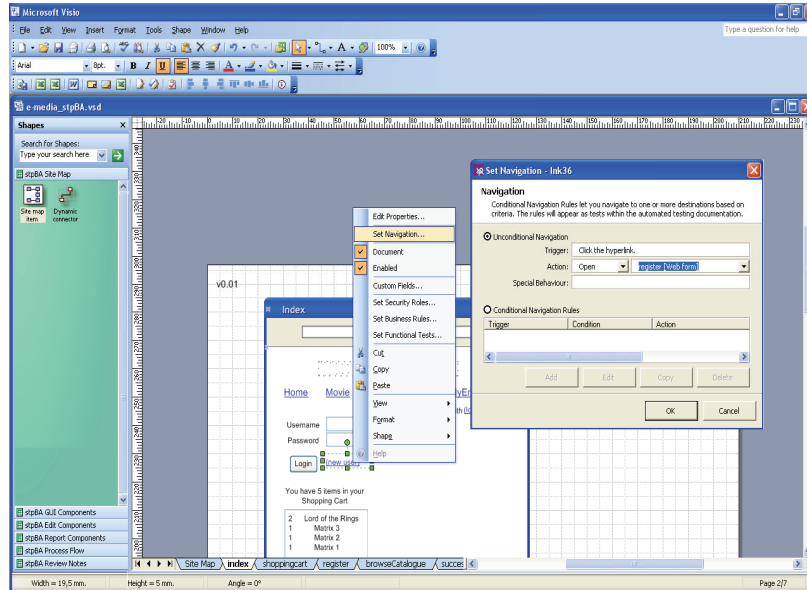


Figure 2-18 An example of UI build with stpBA StoryBoarding

stpBA Storyboarding generates screen flow diagrams, functional specifications and test scripts. Any changes to the storyboard are synchronized with the documentation. The tool is said to generate XHTML output, but in practice this output is only used for the run mode as the output generated is a web page containing a single picture of the prototype. Thus, this tool does not provide any export facilities as the output produced can only be used to validate global navigation between the UIs prototyped.

a. Advantages:

- Benefit from advantages of Microsoft Visio
- Good documentation
- Support for navigation
- Automatic generation of the documentation
- Export to several formats

b. Shortcomings:

- Export (fake html - not reusable export)
- No real code generation
- Run mode

Chapter 2 State of the Art in Informal Design

- ✧ The version proposed (tested on several computers) is bugged, impossible to specify a behavior between screens
- ✧ Cost, require both Microsoft Visio and stpBA Storyboarding

c. Tool Specification

| | | |
|-----------------------|----------------------------------|---|
| Tool Identification | Tool Name | stpBA Storyboarding |
| | Tested Version | - |
| | Last Version | - |
| | Company Name | stpSoft |
| | Contact Person | stpsoft Limited 17 - 21 George Street Croydon Surrey United Kingdom |
| | URL | http://www.stpsoft.co.uk/story/index.html |
| | Licensing | \$ 495 |
| Install and First Use | Required libraries | none |
| | Required software | Microsoft Windows + Microsoft Visio |
| | Recommended hardware | 500 megahertz (MHz) processor or higher and 256 megabyte (MB) RAM or higher |
| | Install time | Low |
| Tool functions | Shape recognition | none |
| | Shape interpretation | none |
| | Code or specification generators | Word, « XHTML », Excel |
| | Level of fidelity | medium |
| | Navigation editor | none |
| | Preview | none |
| | Pattern manager | Can be added |
| | Usability adviser | no |
| UI Editor | Interaction style | Mouse / keyboard |
| | Layout | Absolute coordinates |
| | Granularity | multi windows |
| | Zooming | Well supported |
| | Hand writing recognition | Not applicable (keyboard) |
| | Editing functions | Well defined |
| | Colors | User defined |
| | Assistance | online help... |
| | Language | English |

Table 2-12 stpBA StoryBoarding Specifications

2.4.11 MockUpScreens

MockUpScreens [Mock] is a wireframe editor that permits the designer to sketch screen mockups and organize them in scenarios. Next, MockUpScreens lets the designer experiment interactively with the end users, and quickly visualize scenarios of the application.

Chapter 2 State of the Art in Informal Design

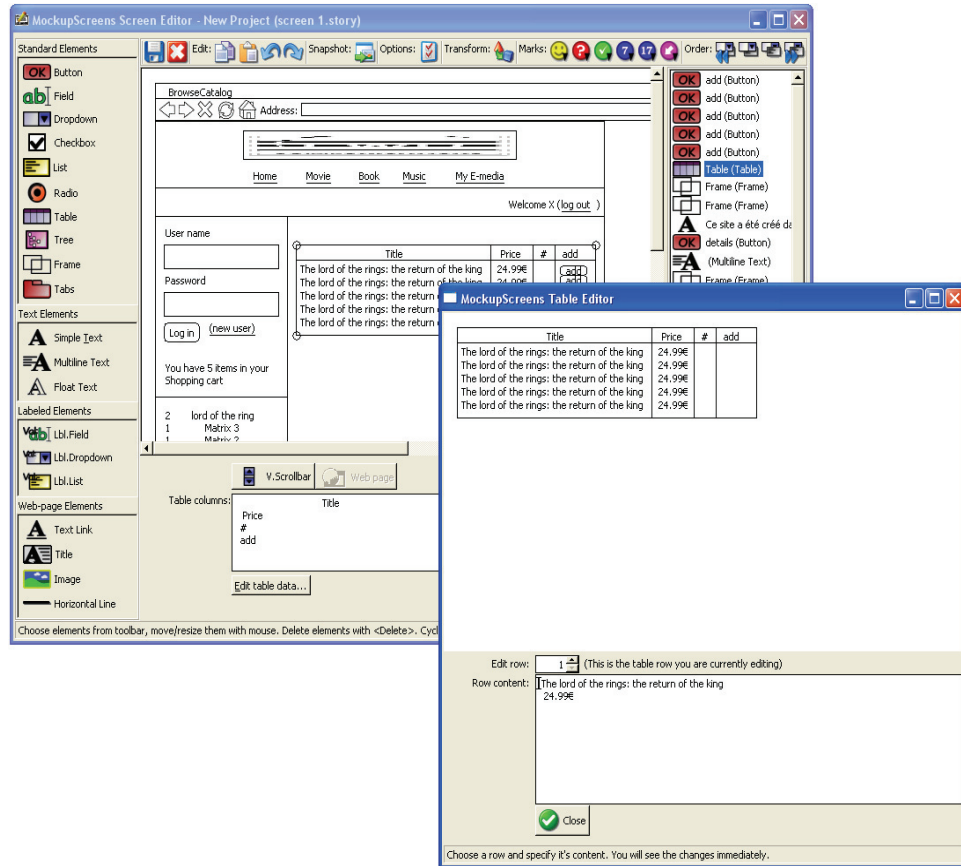


Figure 2-19 An example of UI build with MockupScreens

Through the use of a simple graphical interface, the designers define common screen elements such as buttons, fields, tables, etc. and populate them with data. For web-pages, elements are available such as a web-browser toolbar, links and predefined dummy images. Users can copy and reuse the existing screens, transform screens to and from web pages, move them among scenarios and use predefined templates.

As a result, the designer can choose to export a single screen, scenario or whole project to images for presentation, printing, embedding in documents or similar. Mock-ups purposefully avoid the possibility of being mistaken for the real application screens, but on the other hand there is no reusable output. The html produced only consist in a single picture based web page.

Chapter 2 State of the Art in Informal Design

MockUpScreens integrates very useful and original feature that was not found in other UI prototyping tools so far, it permits to associate element to several pages simultaneously. Then, changes on the common part of the UI is reflected to the whole set.

a. Advantages:

- Easy to discover and to use
- Fast to install, does not require an heavy configuration
- Possibility to transform a widget into another. Designer does not have to remove the first widget before drawing the new one.
- All changes are tracked, so the designer can restore its work to a previous state
- Possibility to annotate the design

b. Shortcomings:

- Widgets cannot be combined together (no groupbox, only text in a table...)
- Table are hard to use
- No export mode (only “fake xml”)
- Some bugs in the UI when resizing the main window
- Difficulties to position widget when dropping them of the window (always appears in top of the interface)
- Simulation consist in a simple slide show, no interaction

c. Tool Specification

| | | |
|----------------------------|-------------------------------------|---|
| Tool Identification | Tool Name | Mockupscreens |
| | Tested Version | 3.9 |
| | Last Version | 3.9 |
| | Company Name | Igor Ješe |
| | Contact Person Brief description | Igor Ješe B.Magovca 30 10000 Zagreb CROATIA igor@mockupscreens.com igor@jeseonline.com |
| | URL | http://www.mockupscreens.com/ |
| | Licensing | 79\$ |
| Installation and First Use | Required libraries | none |
| | Required software | Win98/ME/2000/XP |

Chapter 2 State of the Art in Informal Design

| | | |
|----------------|----------------------------------|-------------------------|
| | Recommended hardware | Small configuration |
| | Install time | Small, automatic |
| Tool functions | Shape recognition | None |
| | Shape interpretation | None |
| | Code or specification generators | None |
| | Level of fidelity | Medium |
| | Navigation editor | Yes |
| | Preview | Yes |
| | Pattern manager | No |
| | Usability adviser | No |
| | | |
| UI Editor | Interaction style | Mouse – keyboard |
| | Layout | Absolute coordinate |
| | Granularity | One single UI at a time |
| | Zooming | yes |
| | Hand writing recognition | No (keyboard input) |
| | Editing functions | Good coverage |
| | Colors | User defined |
| | Assistance | |
| | Language | English |

Table 2-13 MockUpScreens Specifications

2.4.12Axure RP

Axure RP [Axur] is a prototyping tool that enables application designers to create wireframes, flow diagrams, prototypes, and specifications for applications and web sites. The approach proposed by this tool is thus very similar to several of the tools described earlier with MockUpScreens.

The UIs or diagrams are build using a drag and drop approach, while few attributes of the elements can be specified. Axure RP permit to specify more details than MockUpScreens, but far less than Microsoft Visio that permit to specify almost all the attributes that can be associated to a component.

Alike other similar tool, Axure RP permits to specify the global navigation, but it is the only one that supports the local navigation specification. Based on the prototypes, the designer can test the application with the end user thanks to a run mode (export mode). Axure RP covers html code generation, but the code generated cannot be reused since it only consists in a large picture and a set interactive components.

Chapter 2 State of the Art in Informal Design

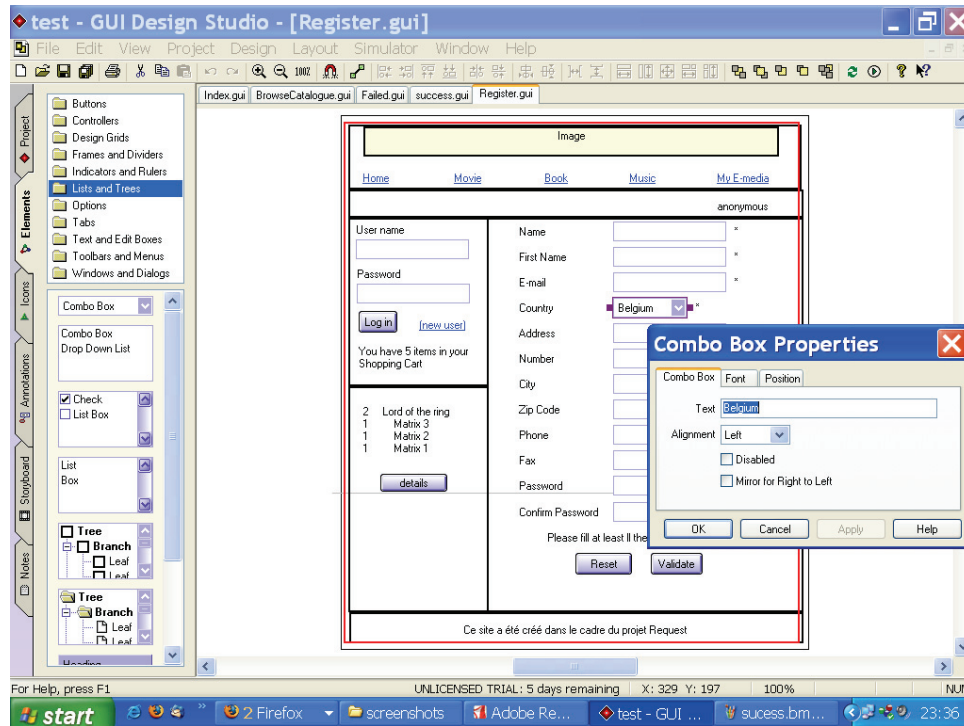


Figure 2-20 An example of UI build with Axure RP

a. Advantages:

- Mature commercial products
- Good documentation
- Support for global and local navigation
- Automatic generation of the documentation
- Export to several formats
- Custom widgets support
- Easy integration with other tools (easy copy from a table from access to the tool)

b. Shortcomings:

- Export (not reusable export)
- Run mode
- Expensive

c. Tool Specification

| | | |
|-----------------------|----------------------------------|--|
| Tool Identification | Tool Name | Axure RP Pro |
| | Tested Version | 4.4.0.741 |
| | Last Version | 4.4.0.741 |
| | Company Name | Axure software solutions, Inc |
| | Contact Person | Axure Software Solutions, Inc. 2667 Camino Del Rio South, Suite 208 San Diego, CA 92108 contactus@axure.com |
| | URL | http://www.axure.com/ |
| | Licensing | \$589 |
| Install and First Use | Required libraries | None |
| | Required software | Microsoft Windows XP/NT/2000 |
| | Recommended hardware | At least 128 Mb memory recommended |
| | Install time | fast |
| Tool functions | Shape recognition | none |
| | Shape interpretation | none |
| | Code or specification generators | Html, Microsoft Word file, csv file |
| | Level of fidelity | high |
| | Navigation editor | yes |
| | Preview | Run-mode |
| | Pattern manager | No |
| | Usability adviser | no |
| UI Editor | Interaction style | Mouse / keyboard |
| | Layout | Absolute coordinates |
| | Granularity | Multi windows |
| | Zooming | Well supported |
| | Hand writing recognition | No |
| | Editing functions | Well defined |
| | Colors | User defined |
| | Assistance | Online and mail |
| | Language | English |

Table 2-14 Axure RP Specifications

2.4.13GUILayout

GUILayout [Blan04] consists in a Java application allowing the designer to draw screens and screen areas, containing other types of regions. This tool is very similar to MockUpScreens, the main difference being the level of granularity of

Chapter 2 State of the Art in Informal Design

the output produced. In GUILayout, each region is assigned to an information type: image, text, title, logo, link, form, navigation, and workspace (e.g., an editor). Oppositely, MockUpScreens permit to specify the UI deeper into details. Moreover, this tool does not provide any support for the navigation.

However, this simple application is very easy to use and permit to produces rough CSS sheets as output or a PhotoShop image.

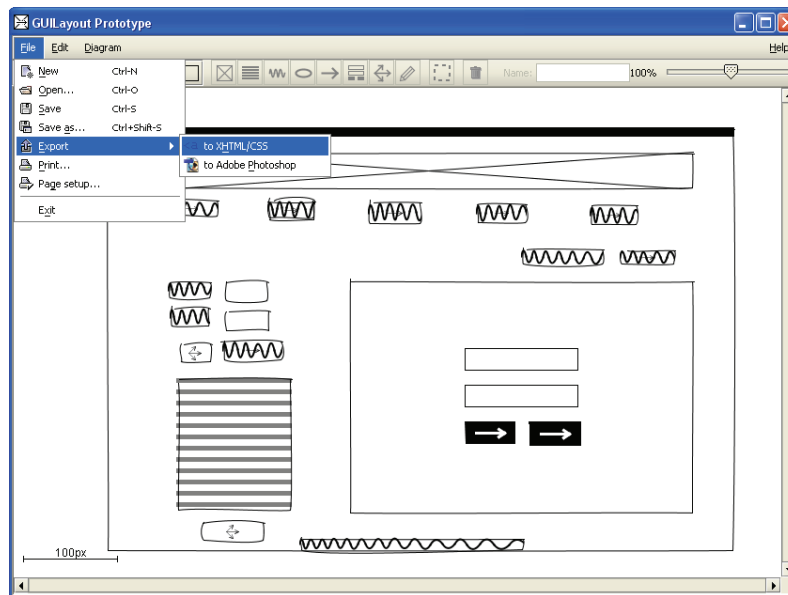


Figure 2-21 An example of UI build with GUILayout

a. Advantages:

- Free
- Easy to use
- Generate html as output

b. Shortcomings:

- Does not produce any re-usable output

c. Tool Specification

| | | |
|---------------------|----------------|------------|
| Tool Identification | Tool Name | GUI Layout |
| | Tested Version | 1.0 |
| | Last Version | 1.0 |

Chapter 2 State of the Art in Informal Design

| | | |
|-----------------------|----------------------------------|---|
| | Company Name | University of Applied Sciences in Furtwangen |
| | Contact Person | Kai Blankenhorn |
| | URL | http://www.bitfolge.de/pubs/thesis/ |
| | Licensing | - |
| Install and First Use | Required libraries | Java web start |
| | Required software | Java 1.4.2 |
| | Recommended hardware | Small configuration |
| | Install time | Very fast |
| Tool functions | Shape recognition | none |
| | Shape interpretation | none |
| | Code or specification generators | Xhtml and psd |
| | Level of fidelity | low |
| | Navigation editor | yes |
| | Preview | Based on xhtml generation |
| | Pattern manager | No |
| | Usability adviser | Ni |
| UI Editor | Interaction style | Mouse / keyboard |
| | Layout | absolute |
| | Granularity | Zooming from a single screen to complete overview |
| | Zooming | yes |
| | Hand writing recognition | No |
| | Editing functions | No undo/ redo but other editing functionalities are present |
| | Colors | default |
| | Assistance | No |
| | Language | English |

Table 2-15 GUILayout Specifications

2.4.14 EasyPrototype

The main principle in EasyPrototype [Easy] consists in letting designers assemble existing paper sketches, whiteboard photos or screenshots into dynamic HTML simulations. The idea is thus to create a UI prototype with some dynamic behavior, but not one that can possibly be mistaken for finished software, and then use it as a tool to drive to agreement with clients and other stakeholders on a project. Similarly to some of the tools presented earlier, EasyPrototype makes it very simple to get a quick storyboard but doesn't include any extra features.

The workflow consists in acquiring pictures of your user interface, saved as JPG, GIF, or BMP. These might be screenshots of Photoshop or Visio mockups, drawings done in Paint, or scans or digital photos of sketches done paper. As we stated earlier, the main drawback associated to paper prototype lied in its

Chapter 2 State of the Art in Informal Design

interactivity, this tool propose thus to solve this major drawback easily. A simple point-and-click interface lets you add tags each screen, a tag is associated to a description, a name, and can trigger a transition from one page to another. Once the tagging phase is completed, you can convert the project into interactive HTML, and optionally build a RTF document describing everything with screenshots.

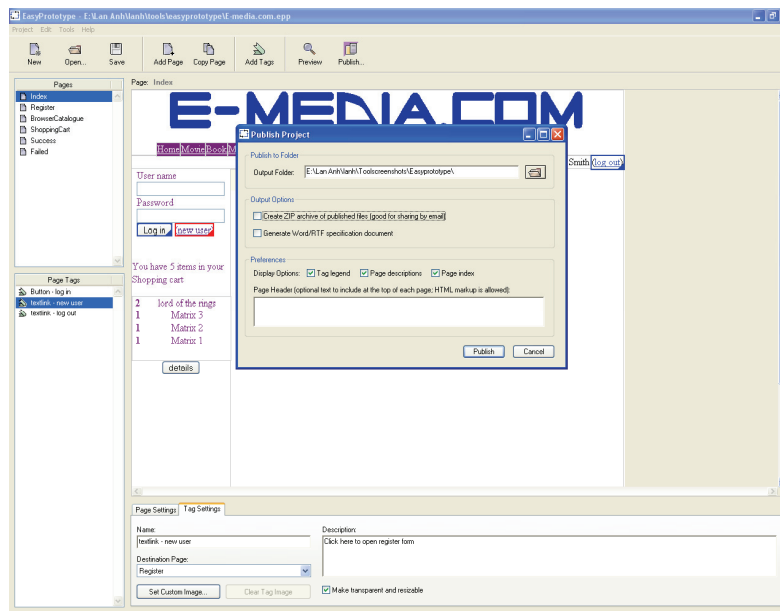


Figure 2-22 An example of UI build with EasyPrototype

a. Advantages:

- Cheap
- Simple solution to the lack of interaction of other tools
- Run-mode in html well developed

b. Shortcomings:

- Limited interest (only navigation)
- Impossible to modify the UI, need to use external tool
- No facilities to acquire paper scan easily

c. Tool Specification

| | | |
|-----------------------|----------------------------------|---|
| Tool Identification | Tool Name | EasyPrototype |
| | Tested Version | 1.5 |
| | Last Version | 1.5 |
| | Company Name | ExtremePlanner Software |
| | Contact Person | ExtremePlanner Software 9811 Kika Court San Diego, CA 92129 |
| | URL | http://www.extremeplanner.com |
| | Licensing | 70\$ |
| Install and First Use | Required libraries | None |
| | Required software | Drawing tool or pictures acquisition software |
| | Recommended hardware | Small configuration |
| | Install time | Low |
| Tool functions | Shape recognition | None |
| | Shape interpretation | None |
| | Code or specification generators | None |
| | Level of fidelity | Any |
| | Navigation editor | Yes, only real functionality |
| | Preview | Yes (run mode in mode) |
| | Pattern manager | No |
| | Usability adviser | No |
| UI Editor | Interaction style | Mouse / keyboard |
| | Layout | n.a. |
| | Granularity | n.a. |
| | Zooming | No |
| | Hand writing recognition | n.a. |
| | Editing functions | Yes, but not really useful |
| | Colors | n.a. |
| | Assistance | n.a. |
| | Language | English |

Table 2-16 EasyPrototype Specifications

2.5 Other Sketch Based Applications

This subsection introduces a set of tools that are not directly linked to the UI prototyping. However, these tools are presented in this section as they address issues and concepts that have a significant interest for this thesis.

2.5.1 EsQUIse

EsQUIse [<http://www.lema.ulg.ac.be/tools/esquise/>] is an interpretative tool for free-hand sketches to support early architectural design. As depicted on fig 3.15, the EsQUIse environment uses pen computer technologies, an electronic pen and a digital tablet-screen, featuring the virtual blank sheet [Juch04]

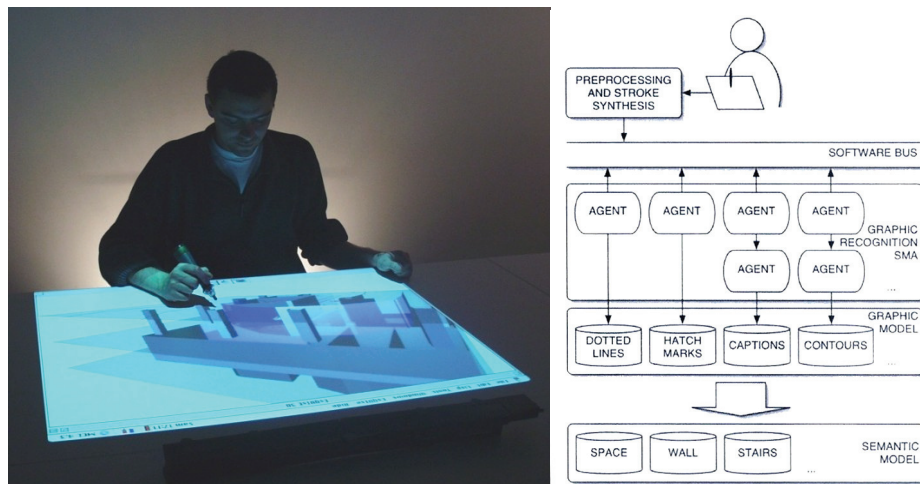


Figure 2-23 The EsQUIse environment and a description of the general system operations

EsQUIse allows the user to graphically describe an architectural plan and to enter the main elements without describing them. Keyboard is never used. No relation is given. No detailed property is feed. Thought, the internal representation of the architectural model knows all the semantic, topological and geometric information to feed classical evaluators of the architectural production.

The working principle of the tool consists in composing the spatial semantic representation of the architectural project in order to feed a computer architectural design environment.

EsQUIsE can then give the geometrical model and the topologic diagram of the design, as needed by basic evaluators and classical tools of architectural production (cost evaluation, future thermal behaviour, 3D models...).

As it can be seen on the general system description showed in fig 3.15, EsQUIsE is based on a multi-agent architecture that extracts characters, words and some symbols recognition which are translated to captions and icons. Then, the graphic model is used by EsQUIsE to construct the architectural model, a veritable “semantic” representation of the building on which the evaluator of the project can work. [Juch04, Mora06]

2.5.2 SketchRead

SketchREAD[Alva04a] is a multi-domain sketch recognition engine capable of recognizing freely hand-drawn diagrammatic sketches. It can be applied to a variety of domains by providing structural descriptions of the shapes in that domain; no training data or programming is necessary. Robustness to the ambiguity and uncertainty inherent in complex, freely-drawn sketches is achieved through the use of context. The system uses context to guide the search for possible interpretations and uses a novel form of dynamically constructed Bayesian networks to evaluate these interpretations. SketchREAD was evaluated on real sketches in two domains— family trees and circuit diagrams—and found that in both domains the use of context to reclassify low-level shapes significantly reduced recognition error over a baseline system that did not reinterpret low-level classifications.

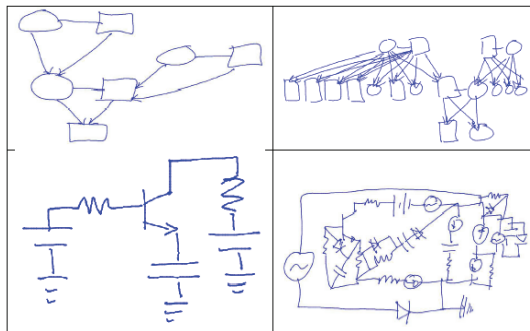


Figure 2-25 An example of UI build with the tiny fingers method

The author pretend in [Alva04b] that SketchREAD can be used to build graphical user interface, but not a single example of such application was found.

Chapter 2 State of the Art in Informal Design

2.6 Summary

| | Paper prototype | Tiny fingers | Silk | Denim | Cabbagh | JavaSketchIt | Freeform 2 | Inkkit | GUI Design studio | Visio | Visio | Storyboarding | MockUpScreens | Azure RP | GUI Layout | EasyPrototype |
|----------------------------------|-----------------|--------------|------|-------|---------|--------------|------------|--------|-------------------|-------|-------|---------------|---------------|----------|------------|---------------|
| Shape recognition | No | No | Yes | Yes | no | no | Yes | Yes | yes | no | no | no | no | no | no | no |
| Shape interpretation | No | No | Yes | Yes | no | no | Yes | Yes | yes | no | no | no | no | no | no | no |
| Code or specification generators | No | No | Yes | Yes | no | no | j | v | j, x | no | no | no | x | x | x, csv | x, psd |
| Level of fidelity | Low | Low | low | low | low | low | low | low | low | high | high | med | med | med | high | low |
| Navigation editor | Yes | no | yes | yes | yes | yes | No | No | yes | yes | yes | no | yes | yes | yes | yes |
| Preview | +- | +- | yes | yes | yes | yes | Yes | Yes | no | yes | yes | no | yes | yes | yes | yes |
| Pattern manager | no | no | no | no | yes | yes | No | No | no | no | no | no | no | no | no | no |
| Usability adviser | No | No | No | No | No | No | No | No | no | no | no | no | no | no | no | no |
| Type of shapes | all | all | ge | all | all | all | p | ge | ge | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. |
| Flexibility of recognition | n.a. | n.a. | low | n.a. | n.a. | n.a. | high | high | high | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. |
| Performance | n.a. | n.a. | | | n.a. | n.a. | high | high | high | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. |
| Conceptual coverage | large | large | low | low | large | large | low | low | (large) | med | med | large | large | large | large | med |
| Number of recognized elements | n.a. | n.a. | 10 | n.a. | n.a. | n.a. | 10 | 10 | N | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. |
| Interpretation mechanism | n.a. | n.a. | g | n.a. | n.a. | n.a. | g | g | g | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. |
| Disambiguation mechanism | n.a. | n.a. | no | n.a. | n.a. | n.a. | yes | yes | no | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. |
| Interpreter extensibility | n.a. | n.a. | no | n.a. | n.a. | n.a. | no | no | yes | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. |
| User adaptive grammar | n.a. | n.a. | no | n.a. | n.a. | n.a. | no | no | yes | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. |
| Hand writing recognition | no | no | no | no | no | no | no | no | yes | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. | no |
| Cut, copy, paste | no | no | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes |
| undo - redo | no | no | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes |
| import - export | no | no | yes | yes | yes | yes | no | no | no | yes | yes | yes | yes | yes | yes | yes |
| multi-windows | Yes | Yes | yes | yes | yes | yes | no | no | yes | yes | yes | yes | yes | yes | yes | yes |

(g) grammar (p) shape primitive (ge) gesture (r) Rubine Algorithmh (o) other (j) Java (x) XHTML (v) Visual Basic

2.7 Requirements for SketchiXML

On basis of the different tools presented throughout this chapter we have identified a set of requirements that should be integrated in a single tool for a better support of the UI prototyping. These requirements are based on a mix between the shortcomings and the advantages identified in the tools described earlier.

- R 1.** *Avoidance of Effort loss.* Some sketching tools only support the sketching activities without producing any re-usable output. So, once the designer and the end user agreed upon a sketch, a contract can be signed between them and the development phase can start from the early design phase, but when the sketch is not transformed, the effort is lost. A better alternative would be to consider a mean to re-use the output in an efficient manner and avoid any effort or time loss.
- R 2.** *Well defined editing functionalities.* The purpose of such tool is to combine the advantages of the paper prototyping and the computer assisted design. As the main advantage of computer assisted design seems to be the possibility to easily move, copy, paste, zoom, undo... these functionalities have to be present in such tool, and must be well defined. Such assertion seems to be very natural, but all the tools do not always propose such functionalities, or just a small subset. Since a design project is likely to involve more than one single UI at a time, it is also necessary for a tool to support multi-windows design. Another significant and obvious functionalities are the import and export functions. It is very important to have the opportunity to save the current work and reopen it later, even on a different computer.
- R 3.** *Language neutrality.* Most of the times, when a sketching tool supports code generation, it is bound to a particular programming language, a particular UI type, a particular computing platform or operating system. So, once an output is produced, it is usually bound to one particular environment, therefore preventing developers to reuse sketches from one case to another, such as for various platforms. As the number of uses is increasing extremely fast, being bound to a specific platform is a clear handicap these days. So, in order to

meet the designer's need, the prototyping tool should provide an output, that is general and context independent. For this purpose we recommend the use of a specification language for UI description. Several specification languages were developed these last years addressing this challenging issue.

- R 4.** *Robust recognition.* If a tool proposes shapes or text recognition, the recognition quality has to be very high, so as to prevent the designer to waste time with misrecognition. Indeed, if a designer has to rewrite the text several time before it is recognized, this feature should be either disabled or improved. Another consideration for text recognition would be to hide the result from the designer during the process, so even if it was not recognized properly, the designer is not tempted to delete and rewrite it.
- R 5.** *Large conceptual coverage.* When considering denim or other similar tools, the conceptual coverage is not a problem since there are not any kinds of constraint on the drawing. For the other kind of tool the situation is different, each tool must specify a set of representations for each widget. These representations can be based on a shapes combination, a single gesture or a mix. Most of the sketching tools providing shape recognition only support a small set of widgets (around 10), preventing to build any complex user interfaces. Moreover, adding new representation is very difficult since most of the tools do not provide any functionality to enrich the grammar. Also, if the designer has the opportunity to add a representation, i.e. a new gesture associated to a widget, adding a new widget is always impossible.
- R 6.** *Recognition and process flexibility.* Most of the sketching tools providing shape recognition try to recognize every stroke drawn (either in batch or real time mode). This prevents the user to represent complex illustration on the future interfaces, such as diagrams, that cannot be represented as widget. Moreover, this constraint also exists for the high-fidelity design tools. For instance, fig 3.19 gives an illustration of a UI that could not be represented with actual low and hi fidelity design tools providing shape recognition.

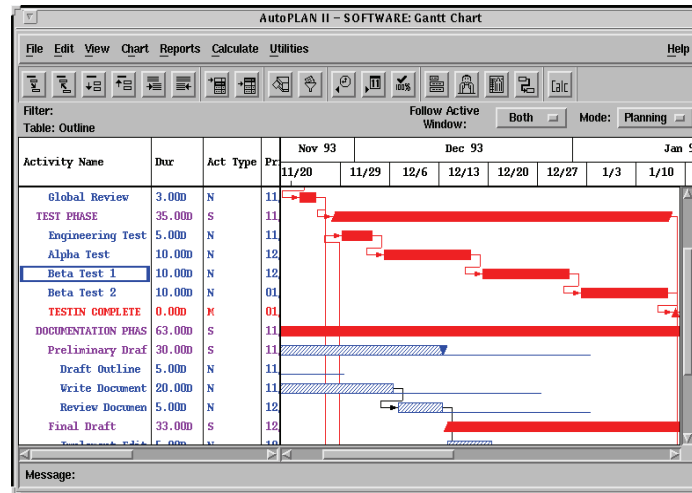


Figure 2-26 An example of UI that cannot be represented with standard UI builder

In addition to the a flexible recognition, the interpretation process should also hold a major role. UI Sketching tools do not allow a lot of flexibility in the sketch process: the user cannot choose when recognition will occur, degrading openness [Sumn97] and when this occurs; it is difficult to return to a previous state. Moreover, depending of the tool, each time a component is recognized, it is sometimes replaced by a stroke of a different color, left as it was drawn, replaced with a smooth representation of the component... According to the authors of these tools, their representation is the most relevant, even if no research tried to confirm such assertions. For our point of view, we should leave this decision to designer and his preferences.

- R 7.** *Design history.* As it can be seen in some professional tools, the use of the design history can be very useful. When prototyping, a designer will tries to explore many design alternatives in a short time, so the designs will evolve very fast. Looking back to the previous steps can be very useful, but such functionalities is rarely supported by the existing tool for rapid prototyping of user interface. In addition to the set of editing functionalities such function would be very useful in this kind of tool.

- R 8.** *Expressive scenario editor.* As stated in the introduction, one of the major drawbacks of the paper prototyping is the difficulty to represent the interaction between the windows. So, we consider that a good prototyping tool should support this feature, since this kind of information can easily be provided by the end user and is important for a global comprehension of the user needs. But all the sketching tools supporting code generation lack of a robust scenario editor.
- R 9.** *Ease of use (naturalness).* The key argument for the development of such tool is the ease of use. Everybody agree on the fact that paper prototype is fast, easy and do not require an extensive background in computer science. To this end, if a tool is supposed to capture the advantage of both computer assisted design and paper prototyping, the main advantage of paper prototype must have a central role in the development of the application. So, the tool must be easy to use, use only natural notation, do not impose any constraints on the sketching... Otherwise a learning curve may prevent the end users from learning how to use the tool and efficiently using it.
- R 10.** *Preview (Run-mode).* One of the drawbacks of the paper-based prototyping is the difficulty to switch from the design phase to a preview or a run mode. The standard approach requires a designer to play the computer and move the window accordingly to the user actions. So, if the tool is equipped with a navigation editor, we can use all the information provided by the end user and build a run mode based on the sketches or the windows interpreted in a specific programming language. Such feature is very interesting since it permit to see how the end users interact with the early prototype.

Based on this list of requirements, we consider having all the elements to unleash the power of informal UI design based on sketches. Through the next section of this thesis we will introduce SketchiXML, which was developed so as so provide an answer to this analysis. SketchiXML lets the designers sketch UIs as easily as on paper combined with all the advantage associated to computer aided design.

Chapter 3 SketchiXML

Development

The content of this chapter is twofold; first we intend to describe the technical aspects of SketchiXML. To this end, the key concepts to be used in the application, the global architecture and the key components are described into details. Second, we present the application itself, explain the working principles and illustrate how the requirements that were identified in the previous chapter are addressed by the application.

3.1 Developing user interfaces for multiple contexts of use

Nowadays, the developers face a new challenge in the design process, as the number of computing platforms is really exploding. Simultaneously the number of programming languages is following the same trend. So, the first subsection introduces a unifying reference framework for multi-target user interface. Based on this framework, the second subsection introduces the UsiXML language, a specifications language for user interfaces description. The last subsection focuses on a specific abstraction layer, as this thesis mainly focuses this layer.

3.1.1 A unifying reference framework for multi-target user interfaces

The unifying reference framework for multi-target user interfaces [Calv03] serves as a reference for classifying UIs supporting multiple targets, or multiple contexts of use in the field of context-aware computing. To this aim, the framework attempts to provide a unified understanding of context-sensitive user interfaces rather than a prescription of various ways or methods of tackling different steps of development.

The framework (Figure 2-8) presents the development life cycle as a set of levels structured with reification relationship going from an abstract level to a concrete one, or going from a concrete level to an abstract one. This is the main reason why this reference framework has been selected. Other reasons are:

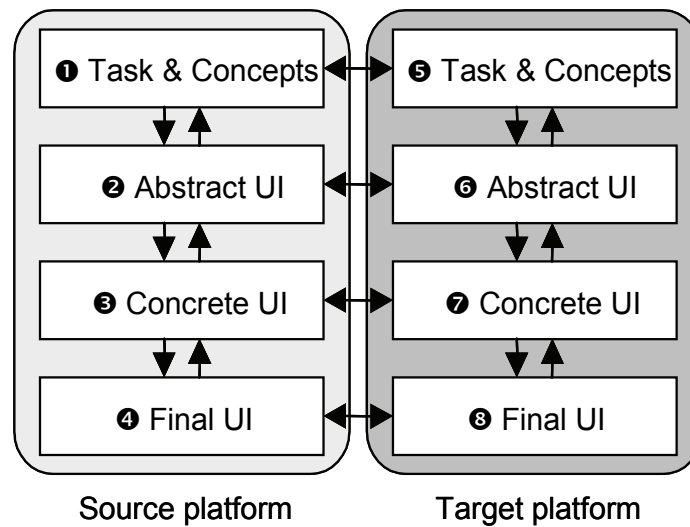


Figure 3-1 The Unifying Reference Framework [calv03].

As presented on the figure, we observe that the four levels of abstraction are:

1. *The Final User Interface (FUI) level*, its rendering materializes how a particular UI coded in one language is rendered depending on the UI toolkit, the window manager and the presentation manager.
2. *The concrete user interface (CUI) level* is assumed to abstract the FUI independently of any computing platform; this level can be further decomposed into two sub-levels: platform-independent CIO and CIO type. For example, a HTML push-button belongs to the type “Graphical 2D push button”. Other members of this category include a Windows push button and XmButton, the OSF/Motif counterpart.
3. *The abstract user interface (AUI) level* is assumed to abstract the CUI independently of any modality of interaction, this level can be further decomposed into two sub-levels: modality-independent AIO and AIO

Chapter 3 SketchiXML Development

type. For example, a software control and a physical control (e.g., a physical button on a control panel or a function key) both belong to the category of control AIO.

4. *The Task & Domain level* describes the various tasks to be carried out by the user in interaction with the system along with the domain-oriented concepts as they are required by these tasks to be performed.

Thanks to this reference framework, understanding and comparing methods and tools is easier, and can be used to express when, where and how a change of context is considered and supported in the context-sensitive user interface thanks to a relationship of translation.

3.1.2 Multi-path UI development: UsiXML

Amongst the different specification languages supporting model based development of user interface, we considered the use of UsiXML for this thesis. Indeed, UsiXML proposes a wide coverage of the presentational aspects of user interface description. Moreover, UsiXML is really open as new concepts can be introduced easily and is in continuous development. Also, UsiXML is based on reference framework, a methodology for user interface in this context is proposed with UsiXML [Limb05]. UsiXML is intended to cover the specification of multiple models involved in UI design such as: task, domain, presentation, dialog, and context of use, which is in turn decomposed into user, platform, and environment. These models are structured according to the four layers of the framework depicted in Figure 2-8: task & concepts (T&C), Abstract User Interface (AUI), Concrete User Interface (CUI), and Final User Interface (FUI).

In order to realize multi-path development, UsiXML proposes an ontology of concepts defining various viewpoints that can be maintained on a UI system. Viewpoints are hierarchically structured depending on their level of abstraction. They describe user tasks, classes of objects, presentational and behavioral aspects of UIs, context of use, and a set of mappings between these representations. [Limb05]

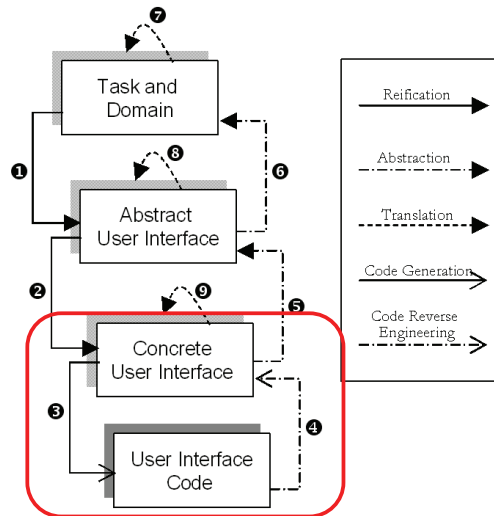


Figure 3-2 Transformation between viewpoints

The downward arrows on Figure 3-2 represent reification steps (forward engineering), from the more abstract to the operational interface. Reification is the transformation of a description (or of a set of descriptions) into a description (or a set of descriptions) whose level of abstraction is lower than that of the source one(s). In the multi-target reference framework, it is the inference process that covers the inference process from high-level abstract descriptions to run-time code.

Upward arrows stand for abstraction steps. This process transforms a description into a description whose semantic content and scope are richer/higher than the content and scope of the initial description content. In the context of reverse engineering, abstraction is the elicitation of descriptions that are more abstract than the descriptions that serve as input to this process. Finally, horizontal arrows correspond to the translation of the interface from one type of platform to another, or more generally, from one context to another.

The underlying mathematical formalism of our ontology being a graph structure (directed, identified, labeled, constrained, and typed graphs), we transform one viewpoint into another by applying conditional graph rewriting rules gathered in graph grammars. These enable us expressing a wide variety of transformational heuristics to express multiple development paths. Ontology and transformations may be stored in an XML format allowing the dissemination, the capitalization, and the consolidation of UI specifications and transformation catalogs [Limb05].

3.1.3 Concrete User Interface

Amongst the four models present in UsiXML, the concrete level is the only level considered in this thesis. The final user interface is also considered in this document, but relies on the interpretation of the CUI via an external tools. The following subsection briefly introduces this layer while the extensive description of the other models can be found in [Limb05]. The subsection is divided into three parts, first the general structure of the concrete layer is introduced, and then the layout mechanism and the behavioral aspect are explained.

a. Description

A CUI is a UI model allowing a specification of an appearance and behavior of a UI with elements that can be perceived by users. A CUI consists of:

- *Modality dependent* i.e., an instance of a CUI addresses a single modality at a time. Two modalities fall in the intended scope of UsiXML: graphical and auditory.
- *Platform independent* i.e., elements populating a CUI realize an abstraction of common languages used to develop UIs.
- *Concrete Interaction Objects* (CIOs) realize an abstraction of widget sets found in popular graphical toolkits (Java AWT/Swing, HTML 4.0, Flash DRK6). A CIO is defined as an entity that users can perceive and/or manipulate (e.g., a push button, a list box, a check box). CIOs are divided into two types: graphical containers (e.g., window, panel, table, cell, dialog box) and graphical individual components (e.g., a button, a text component, a menu, a spin button).

Figure 3-3 propose an example of a user interface specified with the concrete level. Each component is described with a set of attributes specific for each component. UsiXML allows to specify all the attributes that can generally be specify for the components in standard toolkit such as Java/Swing. The structure of the UsiXML file is defined from top to bottom as follow, the head tag contains all the general information associated to the file and its creation. The abstract user interface tags contain the specification for the abstract user interface layer (auiModel). The cuiModel tag is our tag of interest; it contains all the information associated to the concrete level. The context tags describe the context of use

Chapter 3 SketchiXML Development

associated to this user interface. The context contains all the information related to the context associated to this user interface.

```
- <uiModel xsi:schemaLocation="http://www.usixml.org/ http://www.usixml.org/spec/UsiXML-ui_model.xsd"
  id="test_1" name="test" creationDate="2007-03-01T17:57:41.281+01:00" schemaVersion="1.6.4">
  - <head>
    <version modifDate="2007-03-01T17:57:41.281+01:00">1</version>
    <authorName>Adrien</authorName>
  - <comment>
    Generated by GrafiXML 1.2.0 beta build id : 200608210932
    </comment>
  + <comment></comment>
  </head>
  <guiModel id="test-gui_1" name="test-gui"/>
  - <guiModel id="test-cui_1" name="test-cui">
    - <window id="window_component_0" name="window_component_0" width="400" height="350">
      - <gridBagBox id="grid_bag_box_1" name="grid_bag_box_1" gridHeight="17" gridWidth="20">
        - <constraint gridx="1" gridy="2" gridwidth="5" gridheight="1" weightx="1.0" weighty="1.0" fill="both"
          insets="0,0,0,0">
          <button id="button_component_2" name="button_component_2" isVisible="true" isEnabled="true"
            textColor="#000000"/>
          </constraint>
        - <constraint gridx="1" gridy="4" gridwidth="6" gridheight="1" weightx="1.0" weighty="1.0" fill="both"
          insets="0,0,0,0">
          <inputText id="input_text_component_3" name="input_text_component_3" isVisible="true"
            isEnabled="true" textColor="#000000" maxLength="50" numberOfColumns="15"
            isEditable="true"/>
          </constraint>
        - <constraint gridx="1" gridy="6" gridwidth="4" gridheight="1" weightx="1.0" weighty="1.0" fill="both"
          insets="0,0,0,0">
          <outputText id="output_text_component_4" name="output_text_component_4" isVisible="true"
            isEnabled="true" isBold="true" textColor="#000000"/>
          </constraint>
        </gridBagBox>
      </window>
    </guiModel>
  - <contextModel id="test-contextModel_1" name="test-contextModel">
    - <context id="test-context-en_US_1" name="test-context-en_US">
      <userStereotype id="test-sten_US_1" language="en_US" stereotypeName="test-sten_US"/>
      <platform id="test-platform_1" name="test-platform"/>
      <environment id="test-env_1" name="test-env"/>
    </context>
  </contextModel>
</uiModel>
```

Figure 3-3 Example of UsiXML specifications

b. Layout mechanism

The layout mechanism user in the CUI is similar to the layout mechanism that can be found in other user interface description language such as UIML (User Interface Markup Language) [Abra99], programming language such as JAVA /Swing:

A *flowbox* arranges components in a left-to-right flow, much like lines of text in a paragraph. Flow layouts are typically used to arrange buttons in a panel. It will arrange buttons left to right until no more buttons fit on the same line. Each line is centered. The orientation of the layout can be defined as vertical instead of horizontal (see Figure 2-11).



Figure 3-4 Example of flowbox layout

A *borderbox* lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center. Each region may contain no more than one component. The components are laid out according to their preferred sizes and the constraints of the container's size. The NORTH and SOUTH components may be stretched horizontally; the EAST and WEST components may be stretched vertically; the CENTER component may stretch both horizontally and vertically to fill any space left over (see picture 2-12).

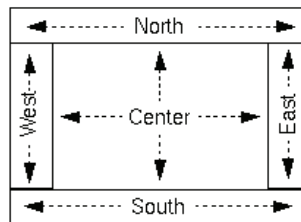


Figure 3-5 Example of borderbox layout

The *gridbox* lays out components in a rectangular grid. The container is divided into equal-sized rectangles, and one component is placed in each rectangle.

| | | | |
|---|----|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | | |

Figure 3-6 Example of gridbox layout

The *GridBagBox* layout aligns components vertically and horizontally, without requiring that the components be of the same size. Each *GridBagBox* object maintains a dynamic, rectangular grid of cells, with each component occupying one or more cells, called its display area.

Each component managed by a *GridBagBox* is associated with a set of constraints. The constraints object specifies where a component's display area should be located on the grid and how the component should be positioned within its display area (see Figure 2-14).

| | | |
|---------|---------|---------|
| Button1 | Button2 | Button3 |
| Button4 | | |
| Button5 | | Button6 |
| Button7 | Button8 | |
| | Button9 | |

Figure 3-7 Example of gridbagbox layout

As it will be presented later, when generating a user interface, the layout mechanism appears as a complex issue. On one hand the designer can choose to specify the layout with absolute coordinates, or to use a more complex layout mechanism that results in a more flexible result.

c. Behavior

In addition to the presentation, the CUI can specify the navigation between the windows or even intra-window. Each of the components can be associated to one or several behaviors. The behavior is based on a Event Condition Action (ECA). The event part specifies the external or internal signal that triggers the invocation of an active rule; the condition part is a condition to be tested for the execution of the action; the action part consists of instructions that either call method, call external programs...

Chapter 3 SketchiXML Development

For instance Figure 3-8 shows a piece of specification from a calculator. The clear button is associated with an action that should clear the display.

```
- <button isEnabled="true" isVisible="true" defaultContent="Clear" id="clear">
- <behavior id="behavClear">
  <event id="evtClear" eventType="release" eventContext="clear"/>
  <action id="actClear">
    <methodCall methodName="clearAll"/>
  </action>
</behavior>
</button>
```

Figure 3-8 Example of a button calling an external method

Figure 3-9 show another example of behavior. This example is taken from a picture viewer application. When an action is performed on the “next” button the current windows should close and the new window should be opened.

```
- <button width="100" height="25" isVisible="true" id="B0" defaultContent="next" name="B0">
- <behavior id="behav0">
  <event id="evt0" eventType="depress" eventContext="B0"/>
  <action id="act0">
    <transition transitionIdRef="Tr03"/>
    <transition transitionIdRef="Tr04"/>
  </action>
</behavior>
</button>

- <graphicalTransition id="Tr04" transitionType="open">
  <source id="B0"/>
  <target id="B1"/>
</graphicalTransition>
```

Figure 3-9 (a) Example of a button calling a transition (b) example of transition.

Based on these specifications, a user interface can be generated in different languages such as XHTML, xul, java, tcl-Tk, flash. Figure 2-17 shows an example of user interface rendered in Flash.



Figure 3-10 Example of Flash calculator based on UsiXML specification [Vand04]

3.2 Agent and Multi-Agent Systems

This section introduces an important concept used later in this dissertation: the agent oriented development. The multi-agent paradigm has appeared during the last decade as a new development paradigm. This trend found its inspiration in the observation of social behavior of humans and insects and considers an agent as a system entity, situated in some environment that is capable of flexible autonomous action in order to meet its design objective [Wool96].

3.2.1 Definition

Three key concepts support the definition of an agent:

- *Situatedness*: an agent receives input from the environment in which it operates and can perform actions, which change the environment in some way;
- *Autonomy*: an agent is able to operate without direct, continuous supervision, it has full control over its own actions;
- *Flexibility*: an agent is not only reactive but also pro-active. Reactivity means that it has perceptions of the world inside which it is acting and reacts to change in quasi real-time fashion. Proactiveness means that behavior is not exclusively reactive but it is also driven by internal goals, i.e., it may take initiative.

Chapter 3 SketchiXML Development

From this, a multi-agent system can be defined as an organization composed of autonomous and proactive agents that interact with each other to achieve common or private goals.

MAS may be either composed of cooperative or competitive agents. In cooperative MAS, the system has a global goal (or set of goals) and the agents that compose the MAS cooperate, possibly by performing diverse tasks, in order to achieve the global goal. This kind of system is typically adapted to perform distributed problem solving. There is a unique high-level goal decomposed recursively into parallel activities to be performed by a set of agents. A good example of such cooperation would be the real-time strategy game Warcraft [War3]. Real-time strategy game usually involves resource gathering, base building, technology development and direct control over individual units. Each unit is an individual agent pursuing one or several individual goals. All the agents are participating to same effort with the same purpose: winning the game. Even if each agent can act individually, they can decide to join their effort, for instance to build a building faster.

In a competitive MAS, each of the component agents has its own set of goals that may or may not meet those of other agents. In this case the MAS is an architecture that allows agents to interact, each one to pursue personal goals and defend its own interests. This kind of systems meets typically engineering requirements of e-commerce, information retrieval applications, web services or peer-to-peer networks. In such environments, every agent generally represents either a client, who wants to obtain some resources or have some service accomplished, or a provider, who wants to sell resources or services at a certain (not necessarily financial) cost. Each agent pursues the goals of the (human or system) actor it represents, and these goals can usually be in conflict.

In order to reason and act in an autonomous way, agents are usually built on rationale models and reasoning strategies that have roots in various disciplines including artificial intelligence, cognitive science, psychology or philosophy. An exhaustive evaluation of these models would be out of the scope of this thesis. Agent models are proliferating; some include learning capabilities, others intelligent agendas based on statistics, others yet are based on genetic algorithms and so on. However, a simple yet powerful and mature model coming from cognitive science and philosophy that has received a great deal of attention, notably in artificial intelligence, is the Belief-Desire-Intention (BDI) model [Brat88]. This approach has been intensively used to study the design rationale of

Chapter 3 SketchiXML Development

agents and is proposed as a keystone model in numerous agent-oriented development environments such as Jack or Jade. The main concepts of the BDI agent model are (except the notion of agent itself we have just explained):

- *Beliefs* that represent the informational state of a BDI agent, that is, what it knows about itself and the world
- *Desires* (or goals) that are its motivational state, that is, what the agent is trying to achieve
- *Intentions* that represent the deliberative state of the agent, that is, which plans the agent has chosen for possible execution

In more detail, a BDI agent has a set of plans, which defines sequences of actions and steps available to achieve a certain goal or react to a specific situation. The agent reacts to events, which are generated by modifications to its beliefs, additions of new goals, or messages arriving from the environment or from another agent. An event may trigger one or more plans; the agent commits to execute one of them, that is, it becomes intention.

Plans are executed one step at a time. A step can query or change the beliefs, performs actions on the external world, and submits new goals. The operations performed by a step may generate new events that, in turn, may start new plans. A plan succeeds when all its steps have been completed; it fails when certain conditions are not met.

3.2.2 Multi-agent systems design pattern

When developing a large and complex application, the reuse of design experience and knowledge from past project is a very important technique [GHJ95, Bush96]. Indeed if a designer has proposed a successful solution for a specific problem, instead of re-thinking the problem from scratch, it is faster to re-use the previous solution.

Based on this observation, practitioners have started to gather design solutions, designs patterns, for many specific issues. A design patterns is thus is a general repeatable solution to a commonly occurring problem in software design. A design pattern is thus not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations. Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved.

Chapter 3 SketchiXML Development

Even if considerable work has been done in software engineering to define software patterns, [GHJ95, Bush96], this work has usually focused on object-oriented systems [Fern01], and hardly ever on the multi-agent systems. Moreover, the proposals of agent patterns [Arid98, Deug99, Hayd99] are not aimed at the design level, but rather at the implementation of lower-level issues like agent communication, information gathering, or connection setup. For instance, the Foundation for Intelligent Physical Agents (FIPA) identified and defined a set of agent interaction protocols that are restricted to communication [Do05].

So as to cover the mismatch between the concepts used by the object-oriented paradigm and other traditional mainstream software engineering approaches and the agent-oriented view [Jenn01, Yu01], [Do05] presents a set high level patterns that are specifically tailored to the development of multi-agent systems using agent-oriented primitives.

Amongst all the patterns presented in [Do05] we only present the Virtual mediator pattern, since this patterns is used in several occasion in this thesis. In the Virtual Mediator pattern, a mediator agent coordinates the cooperation of service providers to satisfy the request of a client. The term “virtual” means that the mediator does not store the answers of the service providers (i.e. they are deleted after the mediator answers the client).

The pattern (3-11) is structured using i^* [Yu95], a graph where each node represents an actor (or system component) and each link between two actors indicates that one actor depends on the other for some goal to be attained. A dependency describes an “agreement” (called *dependum*) between two actors: the *dependor* and the *dependee*. The dependor is the depending actor, and the dependee, the actor who is depended upon. The type of the dependency describes the nature of the agreement. Goal dependencies represent delegation of responsibility for fulfilling a goal; softgoal dependencies are similar to goal dependencies, but their fulfillment cannot be defined precisely; task dependencies are used in situations where the dependee is required.

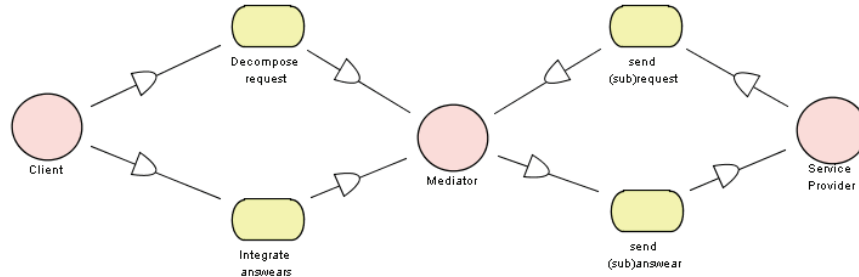


Figure 3-11 Social diagram for the Virtual Mediator pattern

In this pattern, when receiving the client request, the mediator agent is responsible for:

- Decomposing the client request into sub-requests, and then
- Sending each of these sub-requests to the relevant Service Providers

When receiving the answer coming from each service provider, the mediator is responsible for:

- Integrating answers from the Service Providers to formulate final result, and then
- Sending this result back to the Client

3.3 Architectural Description

In order to optimally address the requirements elicited in chapter 2, the choice of the SketchiXML architecture appears of crucial importance. Indeed, so as to meet all the requirements, SketchiXML will have to carry out a large set of interconnected and simultaneous tasks such as providing shape recognition, spatial shape interpretation, handling several kinds of inputs, generating UsiXML specifications, handling complex interaction with the user... Moreover, SketchiXML is likely to be extended in order to integrate additional tools or features, such as a usability adviser, possibly at run time.

In order to illustrate how we address these issues, this section is divided into three subsections. The first subsection presents the general architecture of the application and introduces the key components of the application: the shapes recognizer and the shapes interpreter. The next two subsections present these two components into details.

3.3.1 General architecture

To address the requirements elicited in the previous chapter, we consider a BDI (*Belief-Desire-Intention*) agent-oriented approach [Faul04b] to be appropriate as such architecture allows building robust and flexible applications by distributing the responsibilities among autonomous and cooperating agents. This kind of approach presents the advantage of being more flexible (*R6 – recognition and process flexibility*), modular and robust than traditional architecture including object-oriented ones [Faul04b]. Each critical part of the application is handled by a set of agents cooperating with the others in order to provide the service required according to the designer's requirements.

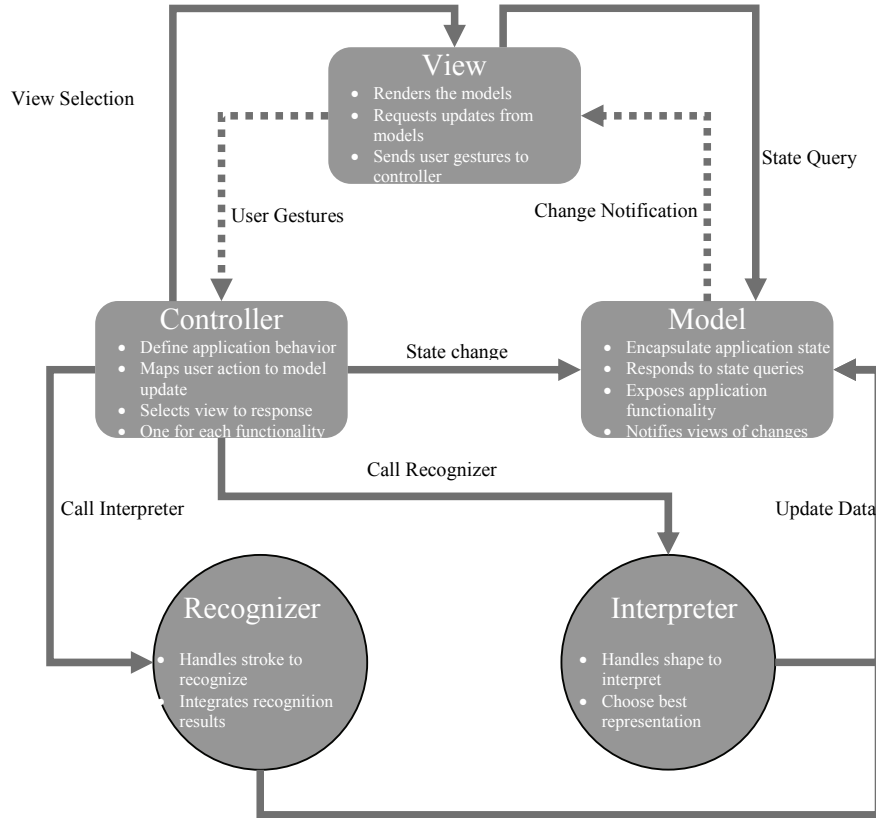


Figure 3-12 SketchiXML global architecture

The architecture of EsQUiSe [Juch04] presented in the state of the art, is based on a similar architecture. The main differences with our approaches consist in the role definitions of the agents. For instance, in [Juch04], the role of the agents is mainly to collect the current drawing and to associate it to a graphic model such

Chapter 3 SketchiXML Development

as: dashed line, heavy line, handwriting... while all the functionality associated to the characterization on the sketch will be provided by a single agent in our application.

The SketchiXML global architecture's is based on a combination of the well known architectural pattern in software engineering: the model-view-controller (MVC) [Beck87], and on a set of multi-agent design patterns [Do05]. The purpose of this model consists in splitting the application into separate layers: presentation (UI), domain, and the data access, so changes to the UI do not impact the rest of application and vice versa. The MVC solves this problem by decoupling the business logic from data presentation and user interaction, by introducing an intermediate component: a controller. Then, for a certain amount of tasks, such as shapes recognition and shape interpretation the controller calls two external modules based on a set of agents. The first module allows to recognize the shapes, gesture and handwriting. The second module is in charge of the interpretation, it takes as input the shapes recognized, widgets identified, text recognized in order to provide other widgets. Except for the shape recognition and interpretation the usage of the MVC architecture does not present any significant interest as it is a well known and simple pattern for application development.

The multi-agent framework that was selected for the development is the "JADE framework". It is based on a middleware that facilitates the development of distributed multi-agent applications based on a peer-to-peer communication architecture. The Intelligence, the Initiative, the information, the resources and the control can be fully distributed on mobile terminals as well as on hosts in the fixed network. The environment can evolve dynamically with agents that appear and disappear in the system according to the needs and the requirements of the context.

JADE is fully developed In Java and is compliant with FIPA specifications [FIPA]. As a consequence a JADE agent can interoperate with other peers not running on the JADE run-time (provided that they comply with the same standard).

This framework presents some clear advantages with regards to other agent platforms as it is a mature product, well documented, totally free and can be integrated in any java development framework. As a comparison, the Jack Framework [Jack] is a very mature agents platform that integrates the BDI model and an editor for the development of agent based application. However, the

Chapter 3 SketchiXML Development

editor provided is less powerful than most on the standard java development environment, thus, a lot of time is wasted due to low performances of the application. Similarly, the Jadex [Jadex] framework permits to develop BDI agent system but require developing the application using intermediate specification in XML. This framework is less mature than the two previous examples and is not appropriated for large project due to the lack of tool support and the time required developing a small application.

The agents are taking part in the interpretation and recognition (see Figure 3-12) processes are described through the following sub sections.

3.3.2 Shape recognition module

As introduced earlier, the SketchiXML recognition mechanism is based on a set of collaborative agents where each agent has a specific role in the process. We have thus developed a specific set of agents for the shape recognition process. A minimum of four agents are participating in this process, three agents are providing the shape recognition for the shapes primitives; handwriting and the gestures, a fourth agent is dedicated to coordination and the integration of the result of these three agents.

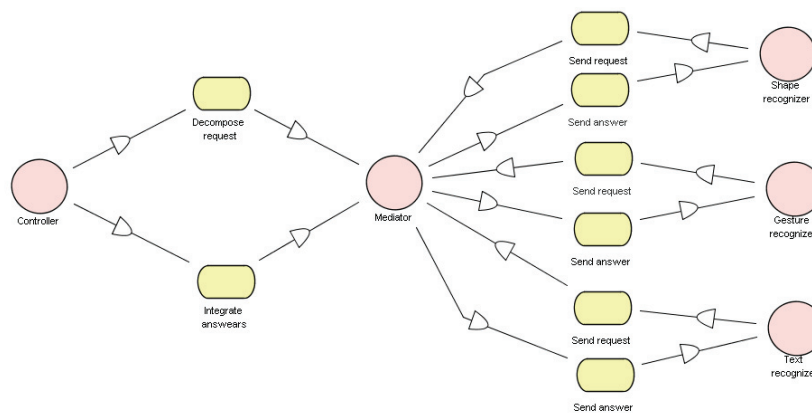


Figure 3-13 Instantiation of the virtual mediator present in Figure 3-11

In this pattern, when receiving the controller (client) request, the mediator agent is responsible for:

Chapter 3 SketchiXML Development

- Decomposing the client request into sub-requests, and then
- Sending each of these sub-requests to the relevant Service Providers

When receiving the answer coming from each service provider, the mediator is responsible for:

- Integrating answers from the Service Providers to formulate final result, and then
- Sending this result back to the Client

Practically, when the mediator receives a new stroke to recognize, it dispatch the information to the different service providers which are likely to provide the appropriate handling. To this end, each service provider has to specify what kind of service is offered. For instance, if a designer sketch a stroke keeping the pen button pressed (command call), the mediator will choose to send the request to the gestures recognizer and to the shapes recognizer. Then, the mediator will integrate the service providers' answers, and choose the more relevant. The mediator can decide to choose the first answer without waiting for all the answers, according to its configuration and if the level of certainty associated with the answer is sufficiently high. Another situation would be the handling of a stroke without any characterization by the mediator; in this case the stroke can be handwriting, a gesture or a shape primitive. In this specific situation, the mediator applied an algorithm on the scribble received in order to detect if it is likely to be handwriting. If the result of this test is positive, then only the text recognizer will be invoked, otherwise all the agents will be asked to contribute to the recognition of the stroke. Each agent involved in this process, the mediator included, adapt their behavior accordingly to their beliefs and the environment.

This kind of architecture could be considered as a service oriented architecture (SOA) for several reasons. According to the OASIS [OASIS] consortium and the reference model they developed: “a SOA is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations. The SOA-RM (Service Oriented Architecture – Reference Model) specification bases its definition of SOA around the concept of “needs and capabilities”, where SOA provides a mechanism for matching needs of service consumers with capabilities provided by service providers.” Based on this definition, the use of agent, each providing a specific service could be easily

Chapter 3 SketchiXML Development

interpreted as a service. Thus, considering each agent as a service make sense, but simultaneously the agents are more than a capability offer as they have more than a single capability and own beliefs. In addition to the fulfillment of a specific task, agents will adapt their behavior according to the context of use. Such observation is particularly important for the shape interpretation mechanism explained 1.3.3. Each agent will store a set of information for the interpretation of the new shape to be handled by the agent. So, the architecture depicted in 3-13 can be considered a multi-agent system, where each agent assumes the role of service provider.

The following section introduces the three libraries or techniques used for the shape recognition in the system. Each of these techniques is thus handled by one particular agent.

a. Shape recognition - Cali Library

The first version of SketchiXML was based on a single recognizer: Cali. Based on fuzzy logic; this library allows to recognize multi-stroke sketches of geometric shapes and single-stroke gesture commands with a very high level of precision.

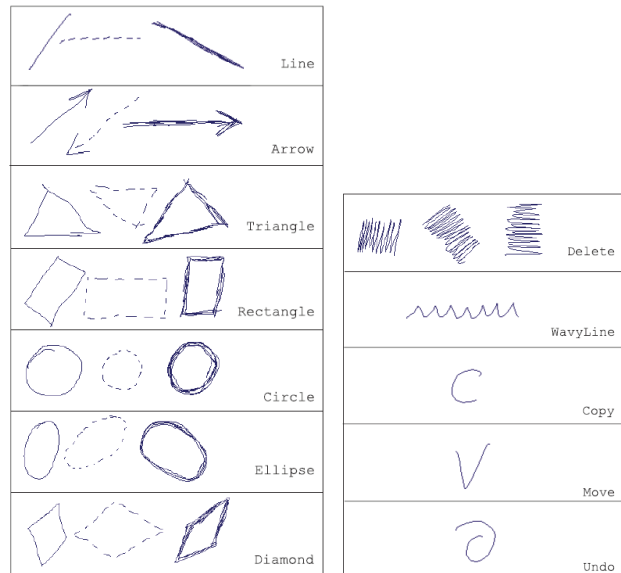


Figure 3-14 Multi-stroke geometric shapes (a) Mono-stroke shapes (b) recognized by Cali library

Chapter 3 SketchiXML Development

The Cali library uses temporal adjacency and global geometric properties of figures to recognize a simple vocabulary of **geometric shapes** drawn in different line styles. The geometric features used (convex hull, largest-area inscribed and smallest-area enclosing polygons, perimeter and area ratios) are invariant with rotation and scale of figures. [Fons02].

The recognition rate obtained with the Cali library is very high: **92 %** of shape recognized correctly and 93 % shape is amongst the top three identified shapes. This recognition rate takes into account the difference between the ellipse and circle and the rectangle and diamonds. This recognition rate is even higher if rectangles and diamonds, circles and ellipses are grouped together. Unfortunately this library does not allow to add custom representations neither to add new shape to the grammar.

b. Gesture recognizer

In addition to the shape recognizer based on the CALI library, we have built **a new trainable recognizer** to solve some of the problems of the existing recognizer (not trainable). The main idea of the new sketch recognizer is to divide a hand drawn input into a sequence of line segments with a particular direction and to compare two of these sequences using the so called string edit distance. A similar approach has been successfully suggested in biometric user authentication, e.g. in [Schi06].

Raw Data

The drawing input from a TabletPC, i.e. the information about the pen movement, is available as a sequence of 3-tuples (x_i, y_i, p_i) , where x_i and y_i are the coordinates and p_i is the binary pen pressure. In our environment, the coordinates are available in units of screen pixels; the binary pressure is set to 1, if the pen tip is touching the drawing surface and set to 0, if the pen is lifted. While using the mouse instead of pen as drawing input device, the pen-down is simulated by pressing the left button.

Feature Extraction

The features to be extracted from the raw data are based on the idea, described in [Free74]. The drawing plane is superimposed with a grid and the freehand drawing input is quantized with respect to the grid nodes (Figure 3-15). Each grid

node has eight adjacent grid nodes and for each pair of adjacent nodes one out of eight directions can be given. So, from the sequence of successive grid nodes, a sequence of directions can be derived. This sequence can be coded using an alphabet $\{0-7\}$, each value representing one direction. This approach was first presented by Freeman in 1974 [Fre74], where it was used for a compressed storage of line drawings. We use the sequence-like representation as our basis for sketch recognition, because it is a short description and location invariant description of complex drawing inputs. For each raw sampling point (x_i, y_i) ($i \in [1, \dots, n]$ for a sequence of n raw sampling points) that closest grid node (qx_i, qy_i) is selected by the following equations:

$$qx_i = \text{round}(x_i / w_g) \text{ and} \\ qy_i = \text{round}(y_i / w_g), \text{ where } w_g \text{ is the grid width (Figure 1).}$$

From the sequence of successive grid nodes (qx_j, qy_j) resulting from sketch input, a string of directions (coded as words out of $\{0 \dots 7\}^*$) of adjacent grid nodes is build. If two or more successive raw sampling points are quantized as the same grid node point, then this grid node appears only once in the sequence. Depending on the grid width w_g and on the distance of the successive raw sampling points, it is possible for the respective grid nodes not to be direct adjacent to each other. In this case the gap can be filled by using the line algorithm of Bresenham [Bres65].

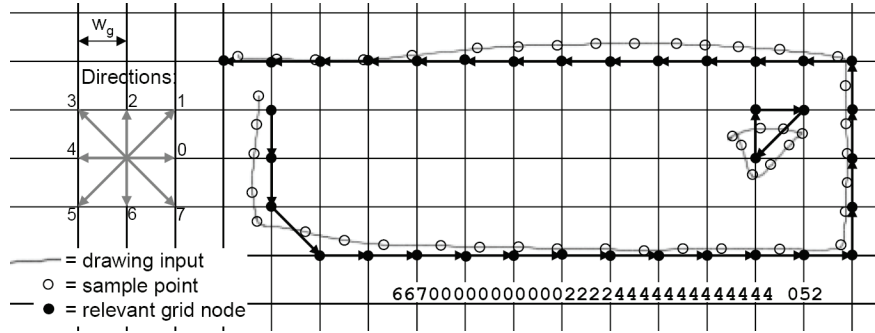


Figure 3-15 Square grid quantization of freehand shapes

The gap between two drawing partitions, i.e. the delay between a pen-up and the subsequent pen-down event can be coded with respect to the relative position of the last grid node (qx_j, qy_j) before the pen-up and the first grid node (qx_{j+1}, qy_{j+1}) after the pen-down. Dependent of the distance and the angle between (qx_j, qy_j)

and $(qxj+1, qyj+1)$, a different coding can be used to indicate the kind of gap. Using this method, it is possible to extract features from hand drawn inputs, which are represented as strings, consisting of codes, which describe the local direction of line segments in chronological order and the characteristic of gaps between drawing partitions.

String Edit Distance

To compare two strings, a common technique is the so called string edit distance, as a measure of their dissimilarity. The idea behind this distance is, to transform one string into another string using the basic character wise operations delete, insert and replace. The minimal number of these operations for the transformation of one string into another one is called the edit distance or Levenshtein distance [Leve66]. The smaller the minimal number of needed edit operations for a transformation from string A to string B, the smaller is the distance between these strings. Instead of only using the number of operations, in some cases it is advantageous to use weights for the different operations. One possibility to determine the edit distance between two strings s and t , with m and n being the respective lengths, is to fill a matrix D of the size $m+1 \times n+1$ as follows [Leve66]:

$$\begin{aligned} D_{0,0} &= 0, \\ D_{i,0} &= D_{i-1,0} + w_D(s_i), \\ D_{0,j} &= D_{0,j-1} + w_I(t_j) \text{ and} \\ D_{i,j} &= \min \{ D_{i-1,j} + w_D(s_i), D_{i,j-1} + w_I(t_j), D_{i-1,j-1} + w_R(s_i, t_j) \} \end{aligned}$$

where s_i and t_j are the i th and j th elements of the strings s and t . $w_D(s_i)$ is the weight for removing operation of a code s_i , $w_I(t_j)$ is the weight for inserting a code t_j and $w_R(s_i, t_j)$ is the weight for replacing a code s_i by t_j . If s_i and t_j are equal, then $w_R(s_i, t_j)$ is zero. The value $D_{m,n}$ is the weighted edit distance of the strings s and t . For a better understanding of the procedure of this computation, we illustrate the resulting matrix in Figure 3-16. It is obvious, that the complexity of the straight forward computation of the edit distance is $O(m \cdot n)$. For each matrix element $D_{i,j}$, the three adjacent elements at the left side and on top (marked in Figure 4.5 by bold border) are required. In practice it can be shown, that the most relevant elements of the matrix D are those around the main diagonal, so the complexity can be reduced, if the grey fields are pre-initialized with an infinite value, so the min-clause of the calculation procedure considers

Chapter 3 SketchiXML Development

stronger the more relevant elements around the main diagonal. Therefore, the computational complexity can be reduced to $O(b \cdot \max\{m, n\})$, where b is a constant factor.

| | | | t_1 | t_2 | t_3 | \dots | \dots | t_n |
|---------|---------|---------|---------|---------------|---------------|-------------|---------|-----------|
| | | 0 | 1 | 2 | 3 | \dots | \dots | n |
| | 0 | 0 | 1 | 2 | 3 | \dots | \dots | \dots |
| s_1 | 1 | \dots | \dots | $D_{i-2,j-2}$ | $D_{i-2,j-1}$ | \dots | | |
| s_2 | 2 | \dots | \dots | $D_{i-1,j-2}$ | $D_{i-1,j-1}$ | $D_{i-1,j}$ | | |
| \dots | \dots | \dots | \dots | \dots | $D_{i,j-1}$ | $D_{i,j}$ | | |
| \dots | \dots | | | | | | | |
| s_m | m | | | | | | | $D_{m,n}$ |

Figure 3-16 Matrix D for edit distance computation

Sketch Recognition using String Edit Distance

As outlined above, the string edit distance can be utilized for the purpose of shape recognition using direction-based feature strings, extracted from handdrawn inputs. The idea is to have a repository, containing a set of reference shapes. For recognition, the unknown shape is compared with all shapes in the repository, i.e. the edit distance between the feature strings of the unknown shape and all reference shapes are calculated. The type of that reference shape, having the smallest edit distance to the unknown shape, is assumed to be the type of the unknown shape. Further, to avoid erroneous recognition of unknown shapes without a representation in the reference repository, a threshold for the maximal allowed edit distance has to be defined.

Due to the nature of string edit distance, the distance value at an average is dependant on the lengths of the strings s and t – the longer the strings, the higher is the average distance value. Therefore a kind of normalization is required. The best solution for considering the lengths m and n in the calculation of edit distance $D_{m,n}$ of two strings s and t is the following:

$$dist(s, t) = D_{m,n} / \max\{m, n\}$$

A second method to normalize the string length impact is to “penalize” large differences in lengths of the two feature strings. It can be assumed, that only if a shape S is different from another shape T , the lengths m and n of the respective feature strings s and t are different. (The inversion is not true – equal lengths of m and n do not imply the equality of the shape types!) By introduction of the string length difference compensation factor the adapted distance could be calculated as follows:

$$\text{dist}(s, t) = d(m, n) \cdot D_{m, n} / \max\{m, n\} \text{ with } d(m, n) = \max\{m, n\} / \min\{m, n\}$$

The effect of $d(m, n)$ is to increase the edit distance by the degree of relative difference of string lengths. Finally, as a third improvement, it is possible to “penalize” the operations *replace*, *insert* and *delete* for the *gap* symbol. The idea is that normally the trained sketches in the repository have the same number of strokes (and consequently the same number of gaps) as the actual drawn shape. So, by using a large weight factor for these “gap operations”, an amount of misrecognitions can be avoided.

The actual recognition of hand drawn inputs can be done by parallel using a set of different grid widths for the quantization while features string extraction. Here, for each single grid width setting, that shape from the reference repository is obtained having the smallest edit distance to the features in the corresponding grid size of the unknown input. So, for a set of different grid widths, a number of decisions for possible types of shape references can be achieved. From this set of decisions a degree of certainty can be derived by dividing the number of matches for each reference type by the number of decisions at all.

c. Handwriting Recognizer

The third agent participating in the recognition process offers the handwriting recognition. To this aim, this agent uses the functionalities found in Microsoft Windows Tablet PC. The Handwriting recognizer pack can be installed on any machine running Microsoft Windows XP, Microsoft Windows 2000 or Microsoft Windows 2003. If the handwriting recognizer is not installed on the computer, the text will be recognized as text, but the content will not be extracted.

3.3.3 Shape interpretation module

Similarly to the shape recognition process, the interpretation is based on a set of collaborative agents where each agent has a specific role in the process. Oppositely to the shape recognition process, the number of agents involved in the process is very high and variable. Indeed in addition to the mediator agent, that holds the same role than the mediator for the shape recognition, we have an agent running for each representation of the widgets representations that can be found in an external editable grammar. The number of agent collaborating in the interpretation process is likely to evolve at run-time. Indeed, the grammar is edited, new agent can be build or destroyed in order to reflect the new composition.

a. Grammar

Each widget is detailed as a precise combination of atomic components and graphical code; graphical codes refer to juxtaposition of components, proximity between components, sequence of components, enclosure of components... The construction of the grammar is detailed in Chapter 4, the following section only details how the grammar is build and used by the application.

For each widget, several representations can be defined, they are defined in an xml grammar that specifies the kind of shapes that should be present in a particular representation to build a given widget. In addition to this list of shapes, the grammar specifies a list of constraints to be applied between the shapes part of the widget. For instance, the list box can be represented with three different representations.

The first representation ($id = 0$) of the ListBox presents a widget that is made of a construct of three shapes: two triangles and a rectangle. Then based on these three shapes we specify a short list of constraints, in this case, the two triangles must be enclosed in the rectangle, in a particular region. Figure 3-18 gives an illustration of the visual representations associated to each of representation specified in the XML grammar shown in Figure 3-17.

```

<widget type="ListBox">
  <representation id="0">
    <constraint id="0" shape1="Triangle_3" shape2="Rectangle_0"
      condition="isInsideInUpperRightCorner" />
    <constraint id="1" shape1="Triangle_4" shape2="Rectangle_0"
      condition="isInsideInLowerRightCorner" />
    <shape id="Rectangle_0" type="Rectangle" />
    <shape id="Triangle_3" type="Triangle" />
    <shape id="Triangle_4" type="Triangle" />
  </representation>
  <representation id="1">
    <constraint id="0" shape1="Triangle_0" shape2="Rectangle_2"
      condition="isInsideInTop" />
    <constraint id="1" shape1="Triangle_1" shape2="Rectangle_2"
      condition="isInsideInBottom" />
    <constraint id="2" shape1="Rectangle_2" shape2="Rectangle_3"
      condition="isInsideOnTheRight" />
    <shape id="Triangle_0" type="Triangle" />
    <shape id="Triangle_1" type="Triangle" />
    <shape id="Rectangle_2" type="Rectangle" />
    <shape id="Rectangle_3" type="Rectangle" />
  </representation>
  <representation id="2">
    <constraint id="0" shape1="Line_0" shape2="ListBox_1" condition="isInside" />
    <constraint id="1" shape1="Line_0" shape2="-" condition="isHorizontal" />
    <shape id="Line_0" type="Line" />
    <shape id="ListBox_1" type="ListBox" />
  </representation>
</widget>

```

Figure 3-17 Extract of the grammar: list box description

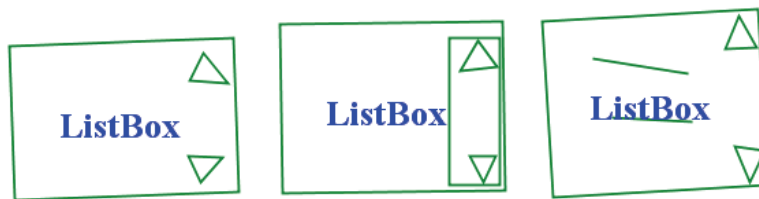


Figure 3-18 Visual representations for the three descriptions associated to the list box in Figure 3-17

In addition to the small set of constraints used in the example provided in Figure 4-6, other constraints also exist:

- areParallel
- cross
- isInsideInLowerRightCorner
- isInsideInTheCenter

Chapter 3 SketchiXML Development

- hasInside
 - hasInsideInLowerRightCorner
 - hasInsideInTheCenter
 - hasInsideInTop
 - hasInsideInUpperRightCorner
 - hasInsideOnTheLeft
 - hasInsideOnTheRight
 - hasPositiveSlope
 - intersect
 - isCrossedBy
 - isHorizontal
 - isInside
 - isInsideInBottom
 - isInsideInTop
 - isInsideInUpperRightCorner
 - isInsideOnTheLeft
 - isInsideOnTheRight
 - isOnTheLeftOf
 - isOnTheRightOf
 - isOnUpperLeftCorner
 - isSmall
 - isSquare
 - isThin
 - isUnder
 - isVertical
 - ...
-

As an example, two shapes will be parallel to each other if the slope of first one is almost similar to the slope of the second one. The accepted margin is dependant of the configuration of the application. When testing if a stroke is inside another, we compute the intersection area between the first and the second, if the proportion of the first one is higher than a specific ratio dependant of the configuration, then the shape is considered to be inside the second. This kind of mechanism is then applied to all the constraints.

All the constraints are hard coded in the application; however they are stored in a java class that can be easily edited. Any changes in the class file are reflected in the complete application. The grammar only specifies the name of the constraint to be applied between the shape, and the access to the method is done thanks to java reflection [Sun98]. Reflection is a feature that is only supported by Java, to my knowledge, which enables dynamic retrieval of classes and data structures by name. The graphical editor for the grammar, presented at the end of this chapter, is also dynamic and adapts the constraints list to the set of constraint present in the java class. If the grammar requires a constraint that is not present in the list of constraints, the representation will never be satisfied.

Two types of constraint exist, the first set on constraint applied to a single shape. The purpose of such constraint is to test a single property of the shape, for instance, the slope of line, the size of a rectangle ... The second type of constraint is used between two shapes and test a specific spatial relation between them.

Chapter 3 SketchiXML Development

The constraints can be applied indifferently between the shapes, the gestures or even the widgets. Figure 3-17 provides a good illustration; the last representation for the ListBox associates a horizontal line to a widget, whatever the composition of the widget is. Indeed, the widget could be built on basis of one of the first two representations, or based on a single gesture associated to this kind of widget.

As the designer is free to define a custom representation for all the components, we cannot predict the geometric properties of the widget. In many situations, we have no other choice than considering an approximation using bounding boxes coupled with Monte Carlo simulations, but when a constraint is applied to geometric shapes we always use the geometric feature.

It is thus possible to build very complex representations involving a large set of shapes and constraints, but as it will be presented in chapter 5, the designer should try to use the fewest shapes and constraints in a single representation. A constraint should be added only if it solves an ambiguity between two distinct representations (R5 - *Large conceptual coverage*).

b. Agents (Widget Agent)

For each representation we create a widget agent that is responsible for the identification of this representation. To this aim, when the agent is created, it logs to the mediator agent, and provide him the list of the shapes required to fulfill its goal. So, when the mediator receives a new shape, it browses its beliefs and extracts the list of all the service providers (widget agent) that can handle this kind of shape.

The widget agents try to build their dedicated widget by testing all the possible shapes combinations and materialize all the relevant combinations as a widget builder. A widget builder, depicted in Figure 3-19, is a widget candidate but still incomplete. For instance if a rectangle is drawn, the two widget agents associated to the two first representations of the ListBox create a new widget builder containing a single shape. Then if a second rectangle is drawn, the widget builder associated to Representation 2 creates a new unfinished widget based on this shape only, and creates a third representation with the first unfinished widget and the new rectangle. Then, the widget agent evaluates if the combination of shapes is valid.

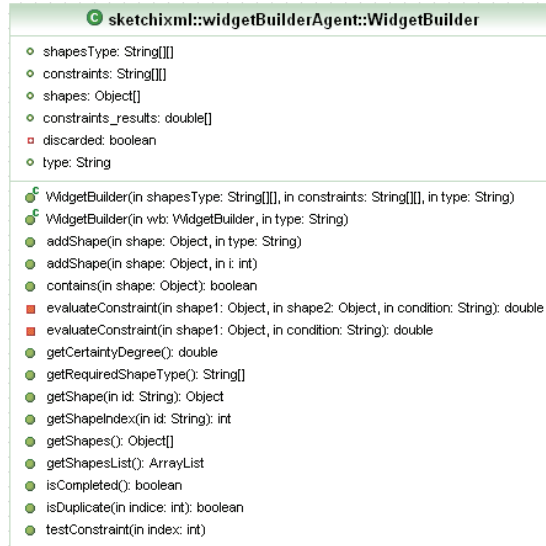


Figure 3-19 Widget Builder class description

In order to be as responsive as possible, the agents optimize their beliefs; when a widget agent receives a new shape, it extract from its beliefs all the representations where just one shape of this kind is missing for the completion of the representation, and provide an answer to the mediator directly. Then, once the agent has provided the results, it tries to combine the shape with the other widget builder and reorganize its beliefs. To this aim, the agent evaluates all the constraint that can be evaluated. A constraint can be evaluated if all the shapes bound to the constraint are already present in the representation, other wise the constraint is ignored until the missing shape is added. All the result from the constraint are store in a table initialized with -1 value. Thus, if the result of the product of all the value present in the constraint result table is negative, we know that some constraint sill require to be evaluated, it the result is 0, we know that at least one constraint does not hold, and the representation can be discarded directly.

Contrary to the architecture of the shape recognition mechanism, the number of agent participating in the process is dynamic. To this end, each widget agent has to subscribe to the mediator in order participate to the process. When subscribing, the agent informs on the type of shapes that can be handled. So, in addition to its role of mediator, the mediator agent also plays a role of yellow-page listing the capabilities of all the agents taking part to the interpretation. Figure 3-20 shows the architecture of the shape interpretation module. Two of the widget agent are instantiated as an example.

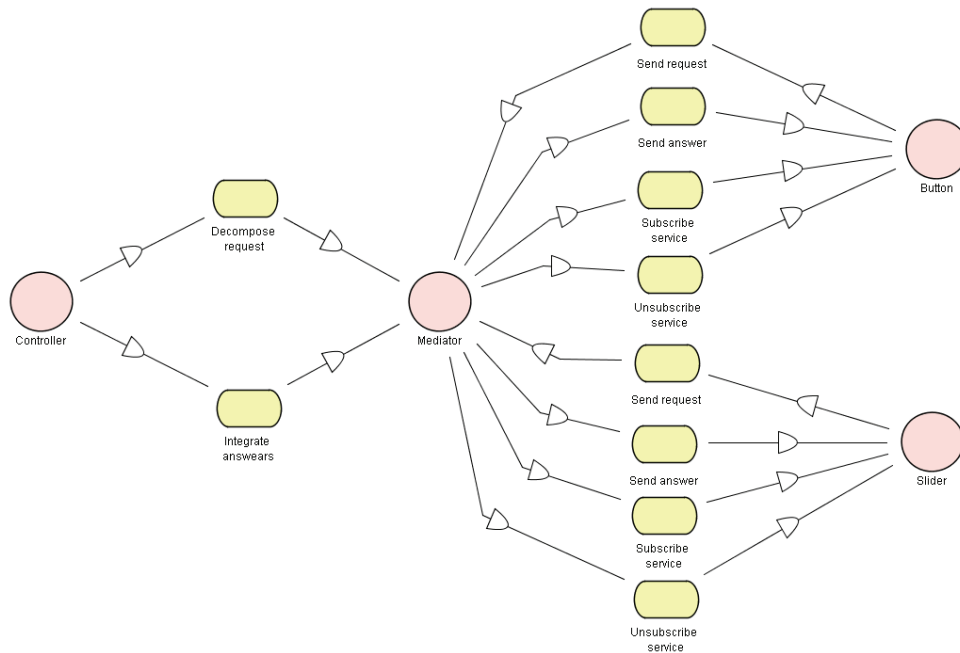


Figure 3-20 Shapes interpretation architecture based on the virtual mediator pattern [do95]

c. Supporting Fidelity Levels

When a widget is recognized by one of the widget agent, the result is then transmitted to the controller that evaluate if the widget should be displayed. Indeed, even if a widget is identified by one of the widget agent, another agent is likely to provide another widget using some of the shape present in the first widget. Once, the controller has decided to display a specific widget according to the results, it creates a Widget object as depicted on Figure 3-21.

A widget contains a set of information such as its type, its size, the set of atomic components used for its creation... Depending of the user preferences, the representations of the view of the set of widget will differ. Indeed, according to user preferences the level of fidelity used for the rendering will be adapted. In order to address this issue, each widget is a subclass from the abstract class Widget. The abstract class Widget proposes a set of method and attributes for describing the widgets. As we can see on Figure 3-21, the class contains the methods “drawHighFidelity” and “drawMediumFidelity” that are invoked to paint

Chapter 3 SketchiXML Development

the widget for the high and medium fidelity. At the lowest level of fidelity (none-fidelity), the rendering of the widget consists only in drawing them as they were initially drawn. Based on the low fidelity level, the widgets recognized are drawn with a different color than the initial sketch and replaces the recognized shape by a smooth representation. The medium and high fidelity levels draw smoother representation that need to be defined for each of the widget individually. That is the reason why these two methods are present in the Widget class.

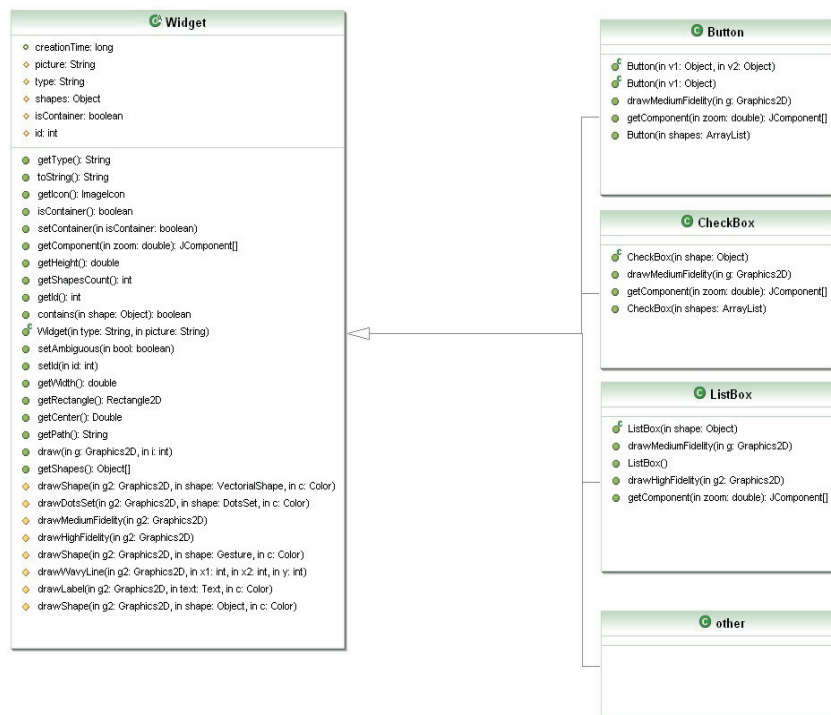


Figure 3-21 Extract of the class diagram for the different widget that are defined by default in SketchiXML

3.4 Presentation of the application

The following section present SketchiXML, the most relevant aspect of the application are detailed in the following, while secondary aspect of the application can be found in the Appendix F.

3.4.1 Parameterize the application

When SketchiXML starts, a dialog box is displayed asking the user to provide a set of parameters for the application.

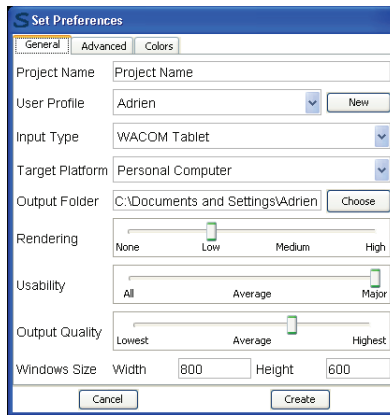


Figure 4-15 SketchiXML parameters dialog box

This set of parameters will drive the low-fidelity prototyping process:

- The project name: determines the name of the current project
- The user profile: each user of the system will have a different configuration and will train the system according to his/ her preferences.
- The input type: determines what kind of input device to use. The selection list displays all the compatible devices found on the computer
- The target platform: determines for which computing platform the UI is prototyped (the actual version of SketchiXML only offers two choices with personal computer and pda, but other platform can be added easily).
- The output folder: allows to choose the location to save the files.
- The rendering level: this slider let the user choose the level of rendering to be used by the application. As it is presented later in the document, the user can choose between four different levels of fidelity ranging from nothing to the high level.

Chapter 3 SketchiXML Development

- The usability level: this slider allows to define how SketchiXML is supposed to interact with the Usability adviser. The usability adviser is a third party application providing real time advice on the design process.
- The output quality: the value selected on this slider will affect the way the sketches are interpreted. When estimating the spatial constraints between the shapes, the result will depend of this value. A lower value increases the speed while a higher value gives a more reliable result.
- The windows size: this two text fields let the user specify the initial size of the windows to be created.

3.4.2 Elements of the SketchiXML Environment

The main window depicted in Figure 3-22 acts as the general manager that initiates the View used by every graphical component. It builds all the different areas and delegates all the graphical representation and event managing jobs to distinct entities, each of them corresponding to a distinct area in the main window.

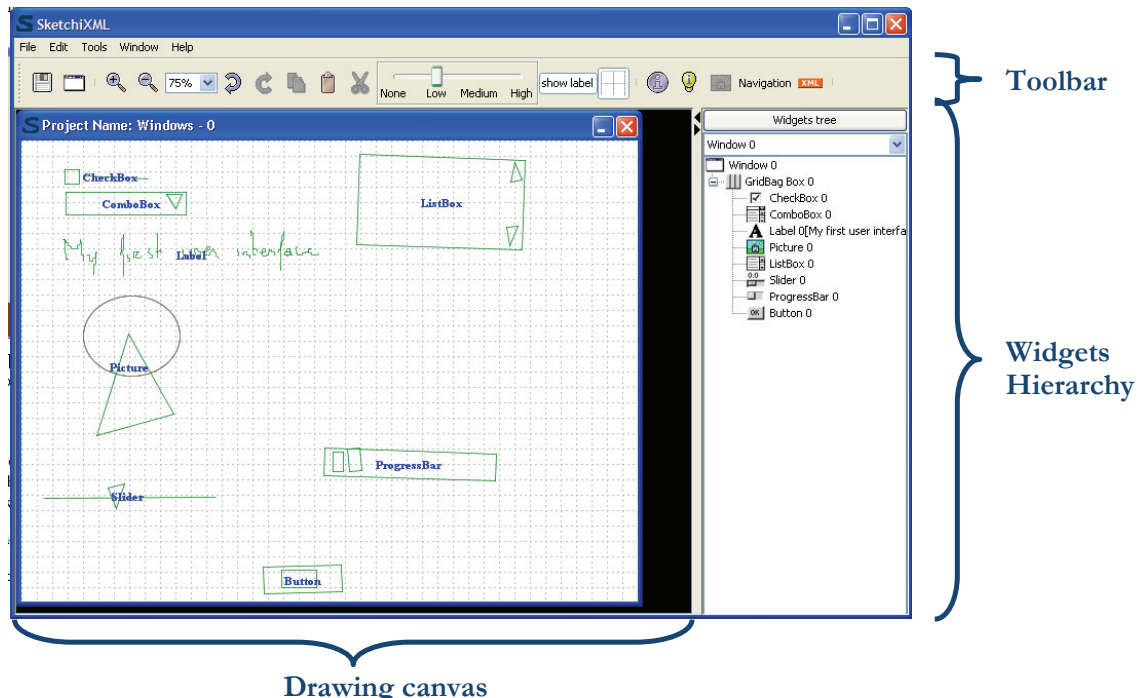


Figure 3-22 SketchiXML main window

Chapter 3 SketchiXML Development

- The drawing canvas is the key component of the application; it is the area where the designer can sketch the user interfaces. Moreover, this surface displays the results of the recognition and interpretation. So, this interface handles a lot of information as input and output. For this purpose, this component required a lot of time to develop robust display strategy. This display strategy involves mainly an efficient repaint of the user window, and tries to avoid disturbing the designer by updating a specific region of the window if the designer is working on that part of the interface.
- The widgets tree on the right of the screen shows the hierarchy of all the recognized widgets.
- The toolbar provide access to all the frequent operation such as copy, paste, cut...
- The menu bar provide access to all the functions including the functions present in the tool bar

3.4.3 Interacting with SketchiXML

As presented on Figure 3-23, there are different possible means of interaction with SketchiXML, you can either use the application by using a standard mouse or by using a pen based device. If your pen-enabled device support custom configuration, you should use this feature and configure the pen as displayed here under. If your pen-enabled device is configured as described bellow, you must choose the mouse as input for SketchiXML.

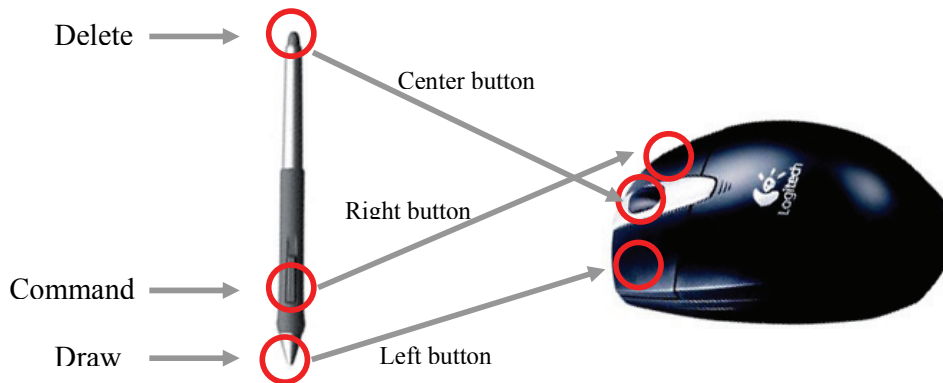


Figure 3-23 functions associated to each button of the mouse or the pen

In both case, three different actions can be done

Chapter 3 SketchiXML Development

You can draw a new widget or part of a widget. If you are using a pen enabled device you just have to draw on the drawing surface. If you are using a mouse you should draw on the drawing area while keeping the right button pressed.

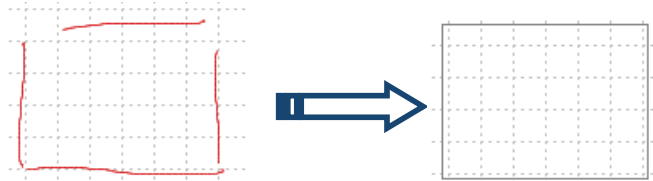
You can delete a widget or a part of a widget. If you are using a pen enabled device you just need to use the eraser present on the back of the pen. If you are using a mouse you should use the center button (most of the time the wheel). Then, draw a stroke on the widgets / shapes you want to remove.

You can call a command using the left button on the mouse or by using the button present of the pen if you are using a pen enabled device.

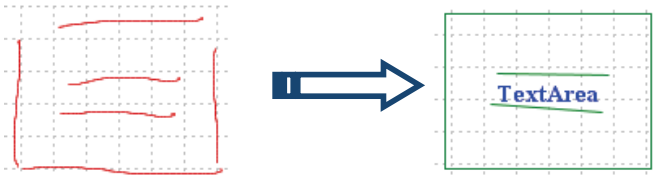
3.4.4 Building Widgets

In order to build widget, the user has to draw the widget using the drawing button of the pen or the mouse. Each widget is made up of a construct of one to several shapes / gestures / widgets and a set of constraints (as depicted earlier).

A multi-strokes rectangle where each stroke is considered to be part of the same shape (delay smaller than 0.5 second between strokes)



A text area based on the rectangle where a delay of half a second was respected between the drawing of the rectangle and both lines



A text area based on the rectangle where delay of half a second was not respected between the drawing of the rectangle and both lines

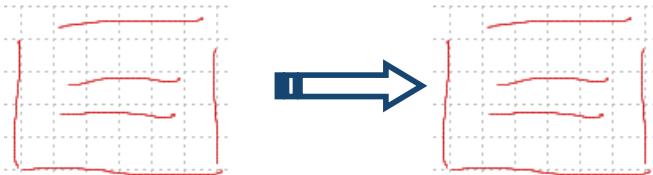


Figure 3-24 Shape recognition and delay between the strokes

Chapter 3 SketchiXML Development

Since the shapes can be multi-stroke, it is compulsory to observe a delay of half a second (default value) between the strokes, otherwise the recognitions engine will not recognize the sketch properly. This delay can be changed on the advanced tab of the dialog box showed at the beginning of the process. Figure 3-24 illustrates this problem, the third sketch is not recognize neither as a rectangle nor a textarea.

3.4.5 Editing functions

As it was stated in the state of the art, SketchiXML is supposed to take the best from paper prototyping and computer assisted prototyping. So, SketchiXML supports all the standard editing functions that can be accessed from the toolbar, the edit menu and by using commands (R2 - *Well defined editing functionalities*).

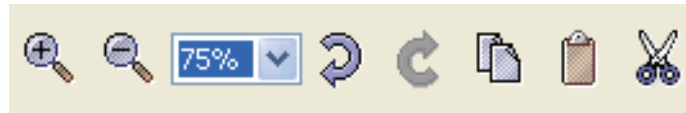


Figure 3-25 SketchiXML edit tool bar

The list of edit function includes the following:

- *Zoom*: the designer is able to choose the zooming level by either using the magnifier icons or the combo box.
- *Undo and redo*: let the designer going back to a previous stage of development. There are no limitations on the number of action to be stored in the undo list.
- *Design memory*: as this tool is supposed to support the prototyping phase, by encouraging the designer to explore many design alternatives, it is important to keep trace of the previous design. To this end, we propose a complete history of all the development steps (R7 - *Design history*).
- *Copy, paste and cut*: require that a part of the UI was selected (cut and copy) or copied (paste). In order to select a part of the user interface, the designer has to surround the area to be selected with a dashed line (at least 5 tokens)

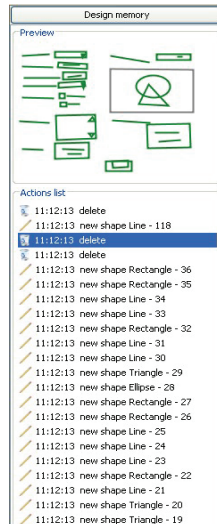


Figure 3-26 Illustration of the design history, all operation are listed and associated with a thumbnail



Figure 3-27 Selection mechanism is SketchiXML; surround the area to copy with a dotted line

3.4.6 Gesture training

If you do not train the system with new gesture, SketchiXML will only be able to recognize a set of shape primitives: circle, ellipse, triangle, rectangle, line, cross... This is sufficient to build most of the simple widget, but in case you want to build complex widget such as a file picker, you may need some extra shapes. To this end, SketchiXML propose a gesture training interface.

The training interface lets you define a set of new representation for shapes, widgets and commands. You can even add new types of shapes and widgets by clicking the add menu. The new shape you will create here will be available in the grammar editor and can be defined as part of a widget.

Chapter 3 SketchiXML Development

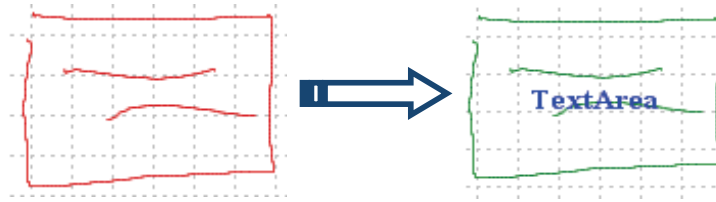


Figure 3-28 training the system with new gesture, that shape that was not recognized previously can be recognized as a gesture and not a combination of shape and constraints

When using the gestures, you are not bound to the half second delay restriction between the strokes. For instance this gesture was not recognized previously, but once a gesture was defined for this widget, it's recognized correctly.

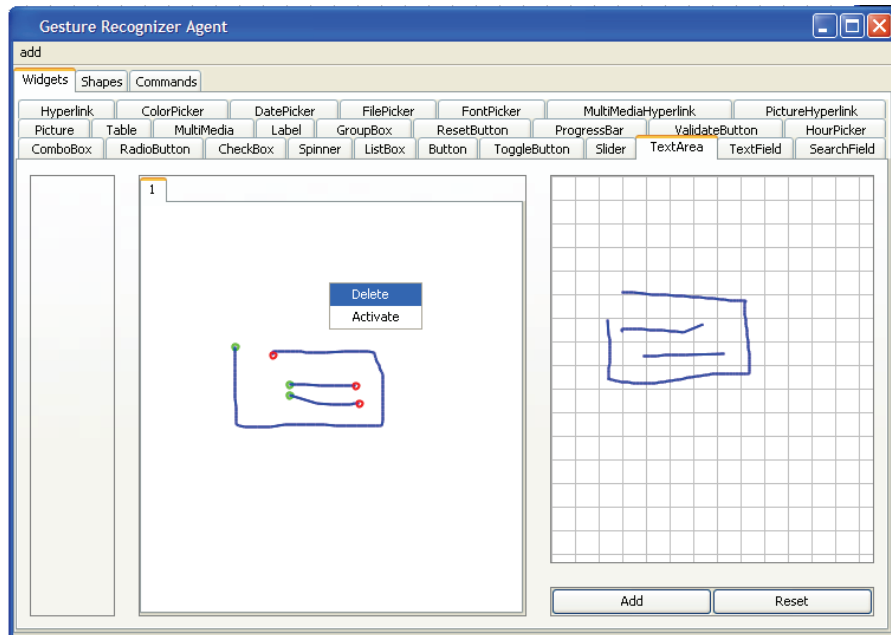


Figure 3-29 gesture recognizer training window, lets the user specifying gestures for widgets, shapes and commands

Contrary to the shape primitive recognition engine, the gesture recognizer is not very flexible. It means that you have to redraw the gesture in a very similar manner that you did when defining the pattern. As an example, if you draw a horizontal line from right to left, it's completely different from a line from left to right. This is the reason why we use profile in SketchiXML, each user has to define its own set of gestures.

Chapter 3 SketchiXML Development

You can specify several gestures for a same widget, shape or command even if it is not recommended. The gesture recognition process is far from being light, so a large number of gestures may affect the performance of the application.

3.4.7 Grammar edition

As presented earlier in this chapter, the grammar is defined in an xml file. In order to edit the grammar easily, we propose a grammar editor that allows to specify new representation for the widgets. For each widget several representations can be added. Each of these representations consists in a group of shapes, widgets or gesture and a list of constraints.

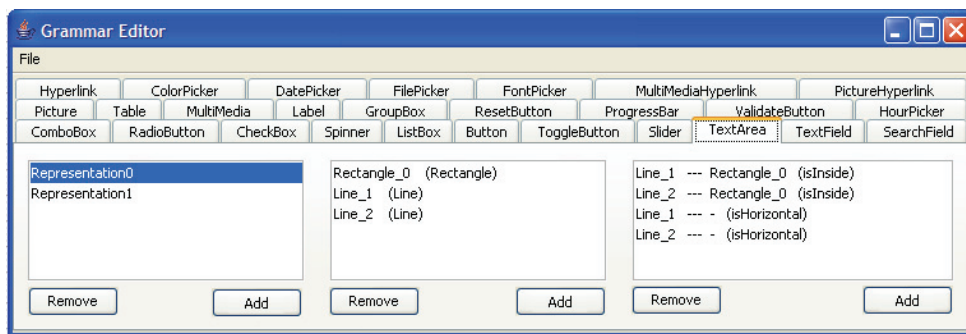


Figure 3-30 Grammar visual editor, permits to edit the representation for the widgets

In order to add a new representation you have to click on the “add” button under the representation list. Then, you need to choose the different shape to be used in the representation. This selection can be done by clicking on the “add” button under the shape list. A list of all the available shape is then displayed.

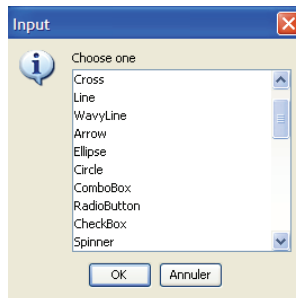


Figure 3-31 Add a shape to a representation

Chapter 3 SketchiXML Development

We can observe that the list contains a set of shape primitives, and also all the different types of widget and all the different types of gestures.

Each time a new shape is added to the shape list, the shape receives a unique name. Once the list of all the shapes to be used in the representation is defined, we can specify the list of constraints. To this aim we have to specify constraint on each individual shape or pair or shapes. To add a constraint, click on the “add” button located under the constraints list, a dialog box will be displayed

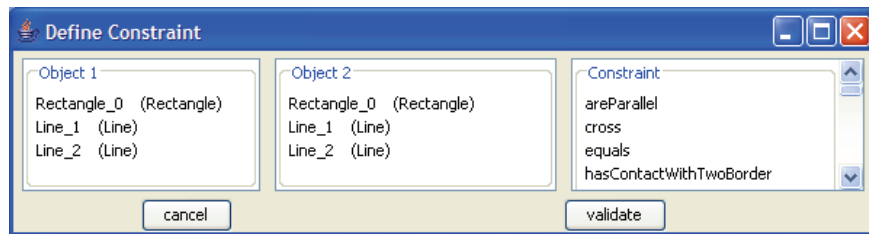


Figure 3-32 Define a constraint between several shapes.

To specify a constraint, select a shape in the “Object 1” list. If the constraint you want to specify does not need a second shape you can directly specify the constraints, otherwise you need to select a second shape in the “Object 2” list. When the constraints is ready, click on the validate button and the constraint is added to the list of constraint associated with this representation.

3.4.8 Level of fidelity

SketchiXML lets the designer choose the rendering level. This level can be chosen at the beginning of the process, but can be changed at any stage of development freely. (R6 - *Recognition and process flexibility*)

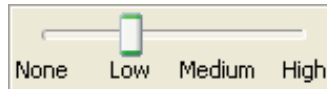


Figure 3-33 Level of fidelity slider, present in the tool bar, allows switching from one level of fidelity to another

The “*none*” level (a) corresponds to a situation where nothing is displayed on the screen. The designer does not get any feed back on his drawing.

The “*low*” level (b) corresponds to a situation all the shapes and widgets that are recognized are displayed on the screen. A shape that is recognized is displayed in

Chapter 3 SketchiXML Development

grey and replaced by its equivalent, and a widget that is recognized is displayed in green with a label in the centre.

The “*medium*” level(c) corresponds to a situation where all the widget that are recognized are replaced by a smoother informal representation of this widget. For the recognized shapes, there are not any changes with the low level of fidelity.

The “*high*” level (d) corresponds to a situation where all the widgets that are recognized are replaced by their corresponding widget in the java Swing toolkit. Other toolkit representation could be added in the future, the reason of this choice is only due to the fact that the tool was developed using this particular toolkit. For the recognized shapes, there are not any changes with the low level of fidelity.

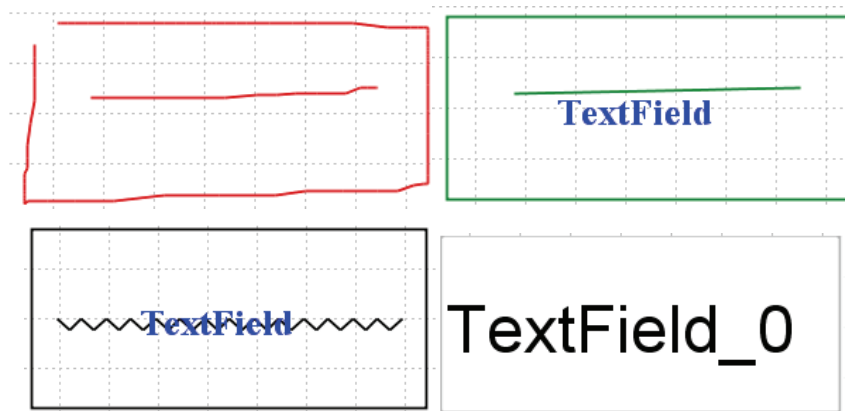


Figure 3-34 (a, b, c, d) The four possible representations for a Text field (from none to high)

3.4.9 Navigation Editor

As stated in the state of the art, a computer assisted prototyping tool should integrate the advantages of the both computer assisted design and paper prototype. One of the drawbacks of the paper prototyping is the difficulty to represent the interaction between the windows. So, we developed a navigation editor that permits to the user to describe the navigation between the windows he just sketched (*R8 - Expressive scenario editor*).

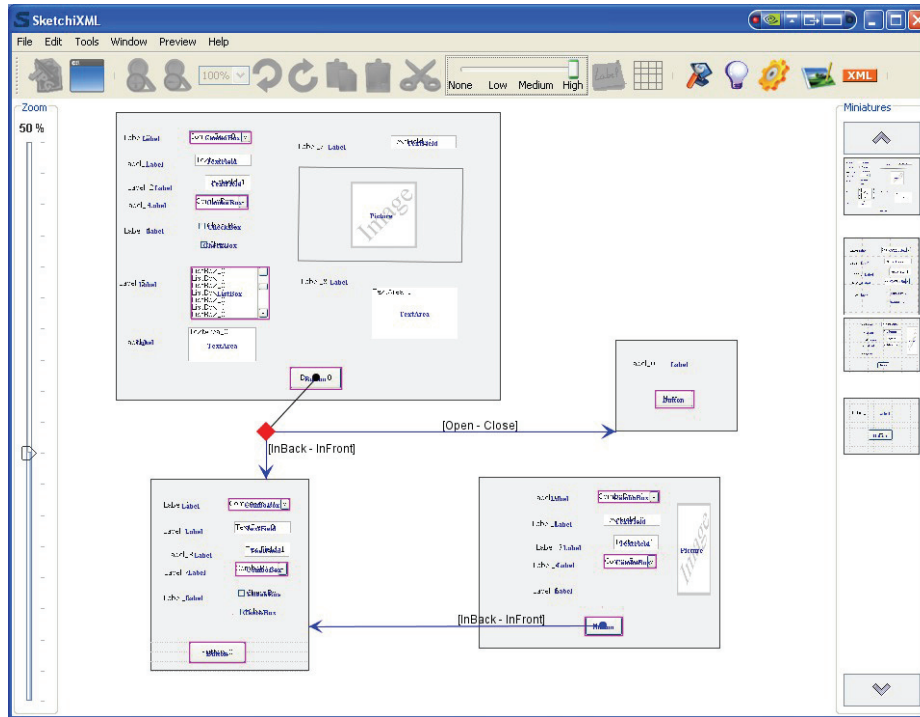


Figure 3-35 SketchiXML navigation editor

The navigation editor only supports a few alternatives for the navigation. Since the purpose of this navigator is to remain intuitive and sketch-based we do not offer the possibility to define complex navigation schema. We mainly support the navigation from one screen to another with several kind of behavior. In order to represent a link from one window to another, the designer just need to draw a line from the component that fire the action. A single component can fire a event that can depend of a condition. In this case, a diamond is displays on the intersection between the transitions.

3.4.10 Preview

One of the drawbacks that was identified in the paper-based prototyping was the difficulty to switch from the design phase to a preview or a run mode. Indeed, the standard approach requires a designer to play the computer and move the window accordingly to the user actions. As we intend to provide a solution for the drawbacks met in both perspectives, we reuse all the information provided in the navigation editor in order to propose a run mode based on the sketches of the

Chapter 3 SketchiXML Development

windows interpreted in a specific programming language. Such feature is very interesting since it permit to see how the end users interact with the early prototype. As it was presented in the navigation editor, one component may trigger an event that executes different actions based on a condition, so when running the prototype, the user is asked to choose the target windows in case of multiple choices. The run-mode rendering is based on the fidelity level used for the design; the example provided in Figure 4.30 illustrates thus the rendering at high level of fidelity. (*R10 – Preview*)

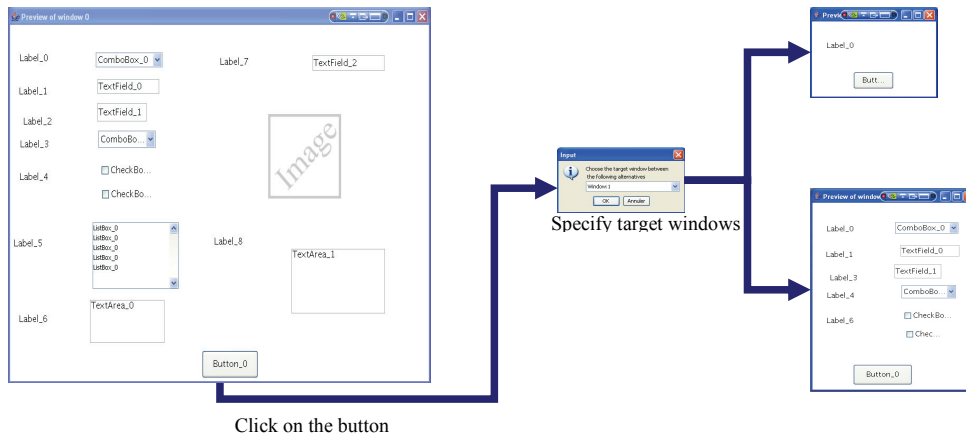


Figure 3-36 Preview mode of the current SketchiXML project

3.4.11 UsiXML output

In order to meet requirements elicited in the state of the art, we needed to address the increasing number context of use issue. So, in order to address this requirement, SketchiXML produces an output, that is general and context independent: UsiXML. (*R3 - Language neutrality*)

As it was presented in the beginning of the chapter, UsiXML contains four abstraction layers. In this case we only produce an output for the concrete UI layer, the corresponding final UI and more abstract specification can be obtained with the combination of other UsiXML compliant tools. UsiXML allows to specify the concrete individual object with a lot of details, in SketchiXML we do not provide value for these attributes as we only generate default values for a subset of the list of attributes

Chapter 3 SketchiXML Development

```
<?xml version="1.0" encoding="UTF-8" ?>
<uiModel id="Project_Name" name="Project_Name" creationDate="2007-03-15T15:02:47.515+01:00"
  schemaVersion="1.6.4" xmlns="http://www.usixml.org">
  <head>
    <version modifDate="2007-03-15T15:02:47.515+01:00" />
    <authorName>Adrien</authorName>
    <comment>This file was generated with SketchiXML</comment>
    <comment>Information on this tool can be found on www.usixml.org</comment>
  </head>
  <guiModel id="Project_Name-cui" name="Project_Name-cui">
    <window id="window_0" name="window_0" isVisible="true" isEnabled="true" width="800" height="599"
      isAlwaysOnTop="false" isResizable="true">
      <gridBagBox id="Box_0" name="Box_0" gridHeight="29" gridWidth="40">
        <constraint gridx="2" gridy="3" gridwidth="3" gridheight="1" weightx="1.0" weighty="1.0" fill="none">
          <outputText id="Label_0" name="Label_0" isVisible="true" isEnabled="true" fgColor="#000000"
            bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
            isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
            visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
        </constraint>
        <constraint gridx="7" gridy="2" gridwidth="7" gridheight="1" weightx="1.0" weighty="1.0" fill="none">
          <inputText id="TextField_0" name="TextField_0" isVisible="true" isEnabled="true" fgColor="#000000"
            bgColor="#ffffff" textColor="#000000" maxLength="100" numberOfColumns="20" numberOfLines="1"
            isPassword="false" isWordWrapped="true" forceWordWrapped="true" isEditable="true" defaultFilter="" />
        </constraint>
        <constraint gridx="2" gridy="6" gridwidth="9" gridheight="1" weightx="1.0" weighty="1.0" fill="none">
          <radioButton id="RadioButton_0" name="RadioButton_0" isVisible="true" isEnabled="true" fgColor="#000000"
            bgColor="#ece9d8" textColor="#000000" defaultState="false" />
        </constraint>
        <constraint gridx="1" gridy="7" gridwidth="9" gridheight="2" weightx="1.0" weighty="1.0" fill="none">
          <comboBox id="ComboBox_0" name="ComboBox_0" isVisible="true" isEnabled="true" textColor="#000000" />
        </constraint>
        <constraint gridx="2" gridy="11" gridwidth="9" gridheight="3" weightx="1.0" weighty="1.0" fill="none">
          <listBox id="ListBox_0" name="ListBox_0" isVisible="true" isEnabled="true" textColor="#000000" />
        </constraint>
        <constraint gridx="3" gridy="17" gridwidth="9" gridheight="5" weightx="1.0" weighty="1.0" fill="none">
          <inputText id="TextArea_0" name="TextArea_0" isVisible="true" isEnabled="true" fgColor="#000000"
            bgColor="#ffffff" textColor="#000000" maxLength="100" numberOfColumns="20" numberOfLines="2"
            isPassword="false" isWordWrapped="true" forceWordWrapped="true" isEditable="true" defaultFilter="" />
        </constraint>
      </gridBagBox>
    </window>
  </guiModel>
  <contextModel id="Project_Name-contextModel_0" name="Project_Name-contextModel">
    <context id="Project_Name-contextModel_0" name="Project_Name-context-en_US">
      <userStereotype id="Project_Name-sten_US_9" language="en_US" stereotypeName="Project_Name-sten_US" />
      <platform id="Project_Name-platform_0" name="Project_Name-platform" />
      <environment id="Project_Name-env_0" name="Project_Name-env" />
    </context>
  </contextModel>
```

Figure 3-37 SketchiXML output, a UsiXML concrete specification

. The solution adopted for the layout follows the same trend, UsiXML offers a complete set of layout, but we only use the Gridbag layout for convenient reason. Indeed, when a designer or a user sketches a UI he expects to obtain a result close to what was sketch. (Wysiwyg revisited: What you sketch is what you get) This assumption does not hold when using other layout as the component are moved

and resized dynamically. Even if the components are also resized and moved with the gridbag layout propose a presentation that is much closer to the sketch. As presented in the beginning of the chapter, each component present is a gridbag is associated to a set of constraints that define its behavior with the surface dedicated to the display of this component. Depending of the widget, these constraints are generated automatically.

3.5 Conclusion

Through this chapter we have introduced two important sections of this thesis. Based on the state of the art provided in the previous chapter we have proposed an innovative approach with SketchiXML.

On one hand, this chapter presents the structure that was adopted for the development of the application, and illustrates how we addressed the issues related to the shape recognition and interpretation.

We have presented the unifying reference framework for multi-target user interfaces development. Then, based on this framework, we have introduced the UsiXML language, a XML compliant syntax, relying on XML schemas to enable a textual representation of any concepts used for user interface construction.

We have presented the agent paradigm and the belief desire intention behavioral model. Additionally, we also presented some design patterns for multi-agent application development. This approach will be used, as a main component, for the development of the prototyping tool presented in this thesis.

On the other hand, the second subsection presents the applications itself and illustrates how the requirements identified in the previous chapter were addressed in a single tool.

Thus, after four years of research and development we are able to propose this tool. The actual version of SketchiXML contains more than 50.000 lines of code without taking into account the code that was generated automatically from XML schema or external jars. Indeed, SketchiXML relies on set of java libraries such as Jade, Jakarta, Apache, Castor... but also on “dll” files for handwriting recognition, shape recognition, connection to the pen device driver...

Chapter 4 A support prototype framework for development methodologies

As stated in the introduction, numerous methodologies already exist for a wide range of approaches. Moreover most of them are already well accepted by the computer science community. The purpose of this chapter is thus to demonstrate how a tool like SketchiXML can be integrated efficiently to the existing methodologies rather than defining a new one. To this end, we start this chapter by a presentation of the UsiXML based tools allowing conducting the prototyping process as depicted in our prototyping reference framework presented in Chapter 2. The set of tools covers almost all the aspects required for a prototyping phase. Based on this tools presentation we propose a general approach to integrate the multi-fidelity levels tools a plug in for most of the classical methodologies. The following presents a selection of methodologies and shows how the guidelines enounced earlier can be applied to this specific methodology.

4.1 Reference framework

Through the second chapter of this thesis, we have introduced a reference framework for the different types of prototyping techniques. The following section presents the set of tools based on UsiXML, providing an effective support for the prototyping phases depicted in Chapter 2.

Chapter 4 A support prototype framework for development methodologies

We start with the high fidelity level since it is supposed being nearest of the expected final interface and permits to see how the various representations of the final interface are transformed in the lower levels of fidelity.

4.1.1 High fidelity prototyping

When a designer uses this level of fidelity, he tries to maximize the proximity between the prototype and the interface to be produced. This means covering a maximum of the aspects of the interface (presentation, navigations global and local), but also the scheduling the functionalities in the time and the space.

For that, it is necessary that the prototype accepts entries from interaction device to be supported (generally, the keyboard and the mouse) in order to treat them and to reflect them in the test of the functionalities. Some prototypes could start with a small data set, not necessarily connected to a data base, so as to ease the development of the prototype.

Most of the time, the developers use the tools that are provided in the integrated development environments, (i.e. Java net Beans). On one hand, these editors are those which support the presentation and the navigation for the platform on which the interactive application is developed; the final user interface can be build by drag and dropping visual object from a palette. Such techniques permits indeed to build user interface very efficiently under certain conditions. On the other hand, these editors restrict the composition of the user interface to a combination of predefined interactive component. So, adding dynamic, not native or not standard components is not possible anymore. To reach this objective, the designer must changer its perspective and code manually the expected component.

Consequently, the user interface builders appear very appropriate for presentation prototype, and particularly for the horizontal prototype while they come to be inappropriate for navigational prototype. Indeed, as soon as the development requires developing a behavior, the designer has to code manually, an expensive activity that should be avoided during a prototyping phase.

Chapter 4 A support prototype framework for development methodologies

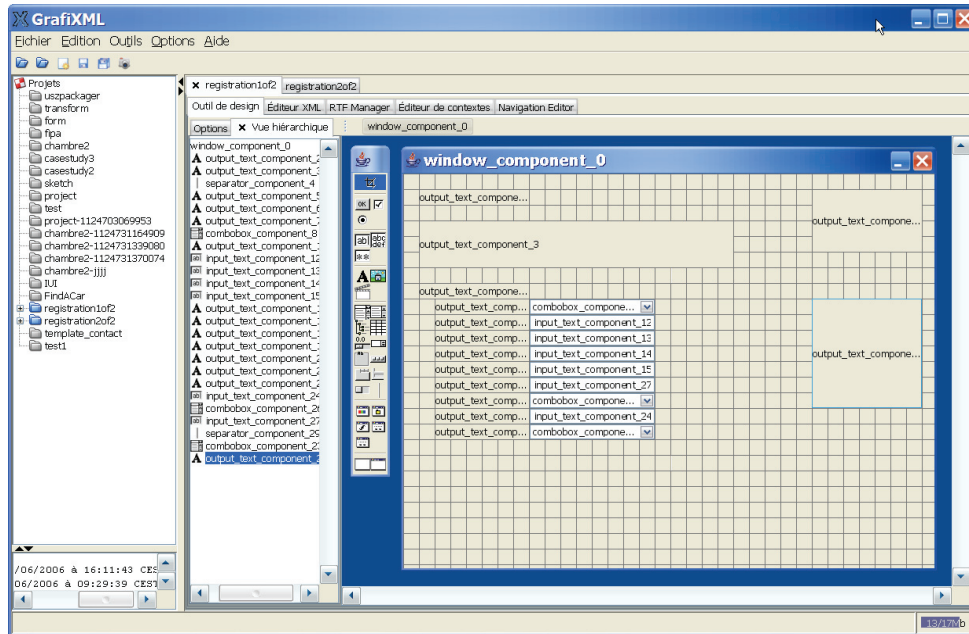


Figure 4-1 GrafiXML main user interface in presentation mode

Due to these limits, some designers prefer to turn to tools not presenting the same gaps, such as the tools known as frontage prototyping tool [Baum96, Berk00, Snyder02]. These tools allow prototyping an interface by building it without coding, so without necessarily having to develop the layer of abstraction and/or the functional core. Hypermedia tools (like HyperCard), computer-assisted presentation tools (like Microsoft PowerPoint, Aldus Persuasion), multi-media authoring tool (as Director Macro-media) were already considered many times as a relevant alternative. Indeed, such tools permit to produce a high level interface with a minimum of navigation, the horizontal prototyping is better supported and vertical or diagonal prototyping are partially supported. The presentation and navigation prototype prototypes are also better supported in this approach. Unfortunately, the effort of production is lost when the designer switch to the development environment planned for the application. Even if such tools generate code, the generated code is not recoverable for the final interface. The developer starts again the development from beginning, with a high fidelity interface.

To answer these challenges, GrafiXML (Figure 4-1), a high fidelity user interface design tool based on the UsiXML language, has been developed. Like other graphical user interface editors, GrafiXML makes possible to represent a graphic

Chapter 4 A support prototype framework for development methodologies

interface by positioning the interactive objects, standard (i.e. a radio button) or not (i.e. a calendar). Instead of automatically or semi-automatically generating the code of the user interface, GrafiXML automatically produces functional and operational specifications written in UsiXML. These specifications can be dependent or independent of a context of use. It is also possible to decline various interfaces for several contexts of use associated with the same project. Figure 4-1 shows a capture of GrafiXML during the capture of the presentation specifications. As this interface can be multi-platform, only a logical representation is posted. In order to obtain a real user interface, the designer can require a preview of the current work (the preview is only based on Java/Swing). Thanks to export functionalities, GrafiXML can generate user interfaces code in several target languages, such as Java, XHTML or XUL [XUL].

The screenshot shows a web browser window titled "Registration (1 of 2) - Windows Internet Explorer". The address bar shows the file path "E:\My Documents\Conférences\ERGO-IA2006\Captures\page.html". The page content is a registration form with a yellow background. It has a title bar, a menu bar (File, Edit, View, Page, Tools), and a status bar (Done, My Computer, 100%).

Registration (1 of 2)

Introduction

In order to be allowed to buy on this web site, you must enter the following informations.

Personal informations

Mr [dropdown]
First name [text field]
Name [text field]
Address [text field]
Second address [text field]
City [text field]
State [dropdown]
Zip/Postal Code [text field]
Country [dropdown]

About this form. Bold fields are required.

Personal Information. Please enter your name and address as they are listed for your debit card, credit card, or bank account.

Figure 4-2 User interface specified in UsiXML rendered with GrafiXML

As an example, Figure 4-2 corresponds to the interface specified in Figure 4-1 generated in XHTML. A style sheet can be associated to the file outside of GrafiXML environment. Logically, GrafiXML can cover simultaneously several languages for the same user interface by specifying the resources which vary according to the user (i.e., the textual resources, the images, the wordings, and the text of contents). Another significant advantage of GrafiXML with regards to other classical editors is its capacity to integrate plug-ins intended to cover wide sets of functionalities. For example, a plug-in exists to transform an existing interface into a PDA version, to evaluate the usability of the user interface, etc...

Chapter 4 A support prototype framework for development methodologies

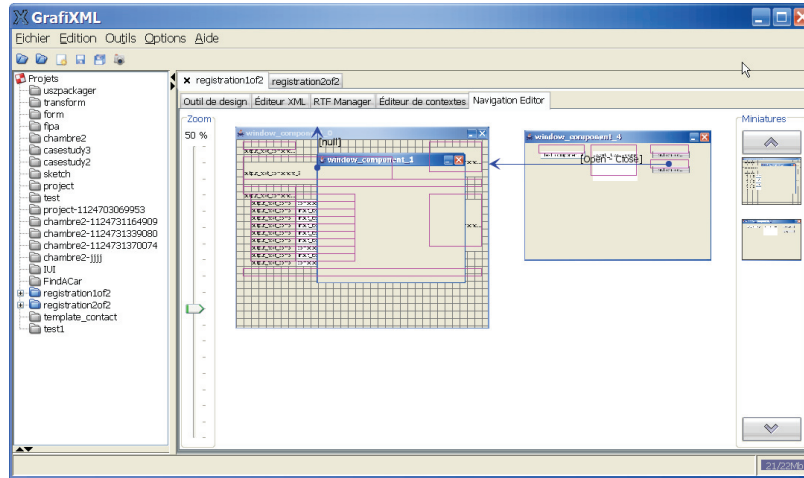


Figure 4-3 Interface GrafiXML main window in navigation edition mode

Eventually, to support navigation prototype, GrafiXML propose a navigation edition mode (Figure 4-3). This navigation editor was presented in the previous chapter as it was initially designed for SketchiXML. Here also, the tool generates the specifications detailed in UsiXML V1.6.4 corresponding to the specified behavior.

GrafiXML currently does not have the capacity to incorporate a not predefined interactive object in UsiXML. Consequently, GrafiXML suffers from the same gap than the standard editor described earlier. We will see however that a partial solution with this problem was founded in prototyping on a lower level of fidelity.

4.1.2 Medium fidelity prototyping

As long as the purpose consists in representing graphically the presentation and/or navigation in GrafiXML, the cost and the time of production seem to remain at least equal to those incurred in a standard user interface editor, the only clear advantage consist in an increased power of expression. This put aside, the specifications written in UsiXML produced by this tool remain always available and make it possible to avoid any effort loss when moving to the next step of the development cycle.

When considering the advanced properties of the presentation or the navigation, the designer can relies on VisiXML, a medium fidelity editor. Such approach makes sense when the designer has a clear representation of the expected result and wants to communicate his idea with a good looking presentation means.

Chapter 4 A support prototype framework for development methodologies

Contrary to GrafiXML where only a logical representation is displayed, the representation used with VisiXML only applies to a single context and give a better understanding of the prototype. This is why it can prove to be judicious to switch on a moderate fidelity level. But, if the designer does not have such a clear overview about the result, he might regret as the time and cost dedicated become prohibitory with regards to the return on investment.

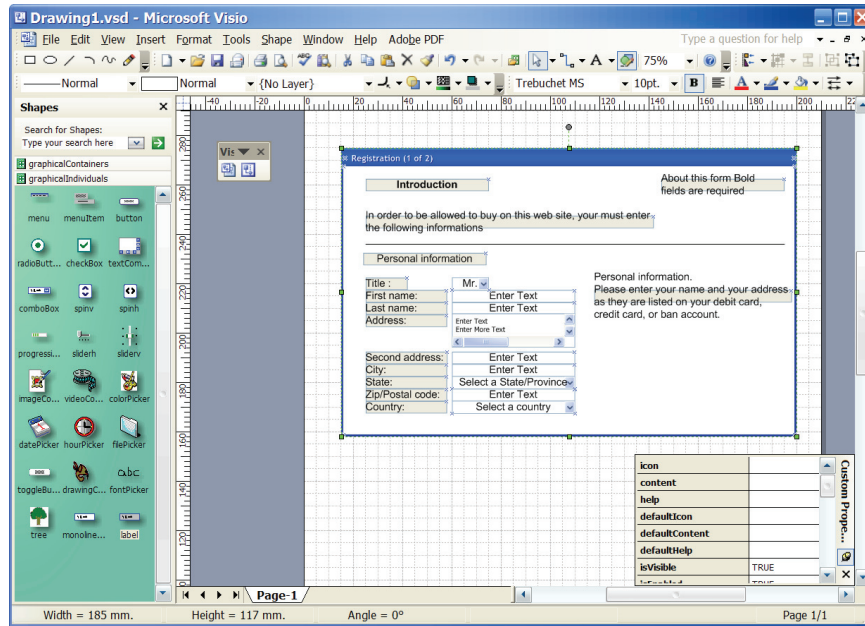


Figure 4-4 Microsoft Visio using VisiXML plug-in

VisiXML (Figure 4-4) is a prototyping tool corresponding to these criteria as it allows considering the visual aspects without requiring programming or additional coding. VisiXML is a plug-in developed within the environment Microsoft Visio Pro, it proposes a set icons of drawing for all the container and component that can be specified in UsiXML (see left part of Figure 4-4). In VisiXML, the role of the designer is reduced to the drawing of the desired interface by selecting the desired containers and individual interactive objects from a list. The fundamental difference between GrafiXML and VisiXML (moderate fidelity) lies in the fact that the output produced is only a set of vectorial drawing that is in principle easily modifiable but whose presentation is related to the target platform, oppositely to GrafiXML where the edition is not related to a particular environment.

Chapter 4 A support prototype framework for development methodologies

As VisiXML is incorporated in the Microsoft Visio environment, the tool takes advantage from all the features propose by this environment: the designer can draw any other basic vectorial object such as decoration, text, figures, drawings. This makes possible to specify elements of presentation which are not defined of standard in UsiXML, but these drawings are lost in export. Once the designer considers the design to be completed, it determines a hierarchy of the objects present in the prototype and generates the specifications in UsiXML. When they are other elements of drawing envisaged by Visio, but not by VisiXML, these elements are safeguarded in the prototype of moderated fidelity, but are lost in the generated specifications. This generates a latent inconsistency between the external representation (the user interface drawn) and the conceptual representation (the user interface specified in UsiXML). On the other hand, the facility of edition is higher, and makes it possible to modify the interface being prototyped faster with fewer details. Indeed, the prototyping is restricted to the physical properties and for the majority of the physical properties having a visual impact, default values are specified in order to minimize the contribution of the originator. Nevertheless, if the designer wishes to specify some of the parameters considered to be important for this level (i.e. font, its size, its color), the properties sheet offers him the possibility to change any of the attributes of a given component. These properties are retained in the generation of the specifications in UsiXML. The major drawback of VisiXML comes when it is a question of exploring a set of design alternative as the representation related to a specific platform does not make it possible to make think that it is about an interface in full evolution: its formal level [Hong01] of details give the impression that the prototype is almost a final interface, and may prevent the end-user to provide an effective feed-back.

So, even if VisiXML restricts the interface building process to a set of vectorial drawings on a given platform, it supports presentation prototyping and global navigation prototyping. Oppositely, the local navigation prototyping is not supported at all, and, contrary to GrafiXML, it will not be upgraded so as to support it.

4.1.3 Low fidelity prototyping

So as to avoid giving the impression of a nearly finished user interface, it is judicious to consider the use of low level of fidelity. When the design is located at a level of low fidelity, the representation of the interface handled by the designer must be as natural as possible to avoid blocking the creative process of

Chapter 4 A support prototype framework for development methodologies

exploratory design. The representation of the interface handled within the editor is thus fundamental to give the impression of un-finished work to the end-user.

Amongst the different reasons, it is interesting to notice that end-users have the same capacity to draw a user interface than a designer: as it is explained in the next chapter, we did not detect a difference statistically significant between the end-users and designers ability to draw a low fidelity user interface. Here is thus a means of really including the final user in prototyping since he can produce a part of the prototype as well.

Also, using a low level of fidelity does not imply that the capacity of the prototype to reveal its advantages and its disadvantages decreases. Actually, it was shown that the number of usability issues identified using a heuristic evaluation provides the same results using a low fidelity or a high fidelity prototype.

Without going into details, the last chapters showed that in SketchiXML we address several fundamental principles govern the prototyping at a low fidelity level:

- *Naturalness*: it is necessary that the user interface being drawn is as natural as possible on one hand. On the other end the drawing constraint must avoid limiting the exploring capability of the user. The results of such process may be thus not immediately similar to the final interface. With SketchiXML the two inputs supported are the handwriting and the sketching. These two expressions means are well known for supporting highly creative design process [Plim04, Snyder02].
- *Non-obtrusion*: it is necessary for the system supporting the draft to be the less obtrusive as possible so as to avoid disturbing the designer during the prototyping phase. The low fidelity representation should not introduce new tasks or actions that are external to the original nature of the activity of prototyping.
- *Continuity*: the support system for the prototyping should support the drawing continuously whatever the nature of the element prototyped (i.e. an interactive object, text, drawing, multi-media contents). The user should not have to change the mode of drawing if an element of different nature must be represented.

Chapter 4 A support prototype framework for development methodologies

- *Recovery*: the effort provided for the draft should be reused in the next step of the development life cycle of the interactive application. In theory, to minimize the costs, the effort supplied during this prototyping, whatever the fidelity level, should be recovered as much as possible in the continuation.

4.1.4 A prototyping framework

Through this thesis, we propose the elements to define a prototyping framework based on the GrafiXML-VisiXML-SketchiXML trilogy. This set of tools support the various levels of fidelity of the user interface prototyping process. Moreover thanks to a common language, we provide support for cross levels navigation as the prototyping trajectories depicted on Figure 4-6 are covered, in theory. Nowadays, no other combination of tools provides such integrated solution.

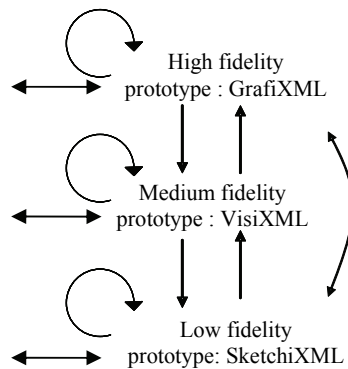


Figure 4-6 prototype development paths based on UsiXML tools

By comparing the tools providing support to prototyping and the different concepts of prototyping presented in Chapter 2, we have identified what tools are appropriate for a horizontal, vertical, diagonal, presentational, navigational total or local prototyping. The experience gained up to now seems to reveal that prototyping at a low fidelity level hold a central role as the naturalness is appreciated: the originator, the end-user itself or the two can collaborate directly to the prototype being designed.

4.2 Application to development methodologies

The purpose of this section is thus to evaluate how this prototyping framework can be integrated in the existing development life cycle. According to [Kend99], a development life cycle refers to the systematic approach analysts take to the analysis and design of information systems. Typically, a system development life cycle is a phased approach to analysis and design which holds that systems are best developed through the use of a specific cycle of analyst and user activities. Each phase is considered as part of the process and not as an individual unit. Instead, several activities can occur simultaneously and activities may be repeated.

Some analysts argued in the past that prototyping should be considered as an alternative to the systems development life cycle. In response to this claim, two concerns are usually evoked. The first concern is the extended time required to go through the development life cycle. As the investment of analyst time increases, the cost of the delivered system rises proportionately. The second concern about using the development life cycle is that user requirements change over time. During the long interval between the time user requirements are analyzed and the finished system is delivered, user requirements are evolving. Thus, because of the extended development cycle, the resulting system may be criticized for inadequately addressing current user information requirements.

It is apparent that the concerns are interrelated, since they both pivot on the time required to complete the development life cycle and the problem of falling out of touch with user requirements during subsequent development phases. If a system is developed in isolation from users (after initial requirements analysis is completed), it will not meet their expectations.

A corollary of the problem of keeping up with user information requirements is the suggestion that users cannot really know what they do or do not want until they see something tangible.

To overcome these problems, some analysts propose that prototyping be used as an alternative to the systems development life cycle. When prototyping is used in this way, the analyst effectively shortens the time between ascertainment of information requirements and delivery of a workable system. Additionally, using prototyping instead of the traditional systems development life cycle might overcome some of the problems of accurately identifying user information

Chapter 4 A support prototype framework for development methodologies

requirements. With a prototype, users can actually see what is possible and how their requirements translate into hardware and software.

The approach advocated by [Kend91] is to use prototyping as a part of the traditional systems development life cycle. In this view, prototyping is considered as an additional, specialized method for ascertaining users' information requirements. This is totally consistent with recent methodologies such as the spiral model that integrate the prototype as part of the process. Such assertion confirm our position on the prototyping process and the need of well defined prototyping tools allowing to build the multiple type of prototypes described in the second chapter of this thesis. In addition to the use of prototype we advocate the use of low fidelity prototyping method as it permit to involve the end user in the conception of the prototype in addition to the evaluation of a prototype built by the designer. Obviously the end user participation is only requested when considering the human-computer interaction. Indeed, for many end users, there are no distinctions between the interface and the system, as the interface is the system. But, as it can be seen in the third section of this chapter, none of the methodologies presented integrate the human computer interaction as part of a general process. Indeed, such domain is often based on psychological theories on the nature of programming and many software engineers seems still reluctant to this kind of specific approach. With SketchiXML and the other user interface builder based on UsiXML we intend to provide an efficient support to most of the methodologies by allowing a strong involvement of the end-users without technical background in computer science.

Naturally, the approach proposed cannot be applied to all the development life cycle effectively; it is best applied to methodologies based on an iterative development life cycle involving user feed back frequently.

Oppositely to the specification-based methodologies, where the requirements consist in a formal and detailed description of system to be, the classical prototype-based methodologies do not provide any detailed specification statements. Instead, the specifications are embodied in the prototype. With our UsiXML-based approach, a part of the information stored within in the prototype is extracted under an xml file containing functional and operational specifications written in UsiXML as an abstract description of the graphical layer of the application.

Chapter 4 A support prototype framework for development methodologies

Our methodology relies on a set of best practices to be applied during a development process based on prototypes or not. These best practices define when to use a prototype, what kind of prototype to use and what tool should be used in order to support the prototyping.

The set of good practices is divided into two groups of recommendations. The first set of recommendation elicits general good practices for the prototyping process that are likely to be integrated in most of the methodologies, and especially user centered methodologies. The following list is not exhaustive as many best practices are already defined in most of the methodologies. For instance, the Extreme Programming presented in the following relies exclusively on a set of best practice. Thus, the following section mainly focuses on the practice dedicated to the prototyping phase.

- *Collect information*, the information is held by various stakeholders, tries to involve at least a representative of each user group in the prototyping process. As [Rett94] claims, in order to conduct the design process efficiently, you have to keep in mind you got to know end user as you are not the user!
- *Develop simple prototype*. Most of the time, a user interface cannot be effectively pre-specified, prototyping is the only manner to evaluate several design alternatives in a short delay so as to understand what the end user had imagined. To this end, a low fidelity prototype can be the communication medium by which requirements can be articulated. The low fidelity prototype can be seen a common language to which the users and developers can relate [Rudd96]. Moreover, from a designer perspective, when building a user interface, many constraints on the layout have to be defined. Defining the constraints based on a mental representation of the user interface is very complicated and be eased with a small sketch of the user interface. So, the designer should always consider the use of a low fidelity prototype when developing a user interface. Moreover, since the output of each type of prototype can be reused, it worth spending enough time to build all the relevant aspects. Indeed, many designer tends to ignore the importance of this phase an consider it as a waste of time as the effort provided is lost
- *Adapt the fidelity level*. Do not hesitate to switch from a level of fidelity to another. As presented in the second chapter of this thesis, each level of fidelity is appropriate for specific tasks. However, do not involve the end

Chapter 4 A support prototype framework for development methodologies

user in prototyping development activities that do not require his participation. Interaction with the end user should only occur during low fidelity prototyping construction or for the evaluation of higher fidelity prototype.

- *Discuss with the end-users.* When requiring end user participation, prepare realistic tasks and scenario in order to stimulate the end user in order to gather as many information as possible. Ask questions to the end-users, much information can be gathered through questioning and debriefing. Indeed, a lot of information that cannot be highlighted on a low fidelity prototype, such as information on non-functional requirement, could be expressed in an indirect manner through discussion.
- *Test interaction between end user and prototype.* Low-Fidelity prototypes generally require a facilitator, who knows the application thoroughly, to demonstrate or to test the application. Interactivity by the user is somewhat restricted. The user is dependent on the facilitator to respond to the user's commands to turn cards or advance screens to simulate the flow of the application. [Rudd96] Based on SketchiXML, the designer can develop the user interfaces together with the end users, but also the global navigation with the set of user interfaces.
- *Avoid polished prototype.* Making the prototype too polished may be counter-productive. Firstly, such prototype might force users to accept it as finished. Secondly, you are likely to hear criticisms about the colors typography or other details. Thirdly, as a designer, if too much time is spend on a prototype, you likely get too attached and fall in love with it.[Rett94] Interacting on basis of a low fidelity prototype is thus more suitable and facilitate the interaction with the end-user as he will be more likely to provide his comments.
- *Iterate.* As explained in the introduction, when the prototype progresses, it goes through several iterations. According to [Rett94], the number of iterations of the design-prototype-test cycle improves the quality of the resulting design. So low-fidelity prototyping is a technique that can dramatically increase quality.

Chapter 4 A support prototype framework for development methodologies

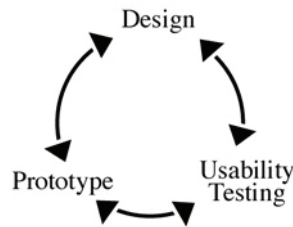


Figure 4-7 Iterations between design – usability testing – prototype is the key

In addition to the general good practices, we propose a set of good practices that are directly linked to the application development life cycle. To this end, we consider the development life cycle into a succession of seven distinct activities. For each of these development steps we detail what kind of support is expected from the UsiXML-based tools when relevant.

- *Planning*. During this first step, the designer establishes the plan for creating the information system. This phase define the system to be developed and present the project scope with a high level of abstraction. This phase should produce the project plan, containing the tasks to be completed,
- *Systems analysis, requirements definition*. Refines project goals into defined functions and operation of the intended application. Analyzes end-user information needs. During this phase, the purpose is completely different from the previous point. Exploratory prototype can provide an effective support to determine user requirements. Such prototype lets the users interacting with the prototype being developed, and permits to elicit new requirements that may have been overlooked. Based on SketchiXML, such prototype, that is generally considered to be a throwaway prototype, can be reused in the later stage of development. Moreover SketchiXML propose amongst other functionalities, to test the current design by animating the prototype. Furthermore, SketchiXML can be adapted so as to responds to the end user expectation, as the result can be interpreted in different means with different types of rendering. At this level, SketchiXML fully achieved is role of communication and interaction means between the designers and stakeholders.
- *Systems design*. Describes desired features and operations in detail, including screen layouts, business rules, process diagrams, pseudocode and other documentation. During this phase, the prototype developed should provide

Chapter 4 A support prototype framework for development methodologies

effective support to evaluate the design of the systems components such as database, algorithm, architecture, human-computer dialogue... Obviously most of these aspects cannot be prototyped with our prototyping framework as these aspects are clearly out of the scope of the framework. However, the human-computer dialog can be specified using the GrafiXML or SketchiXML, even if the local navigation is not yet supported, GrafiXML as a high fidelity editor will permit to define a detailed specification for the navigation local and global. Based on this specification, GrafiXML is then able to generate an interactive prototype, allowing previewing the design and interacts with the system.

- *Development*. This last step consists in the real code of the application. Based on the specification provided by the user or the designer through the different steps and prototyping techniques, GrafiXML generates the corresponding code in the desired language.
- *Testing*. This phase consists in testing the developed system. Test are conducted so as to compare the effective performance with the expected results.
- *Implementation*. The system is put into use. A user guide should be developed and the end users are trained to the use of the system.
- *Maintenance*. This last phase consist in keeping the system up to date.

4.2.1 The waterfall model

The waterfall development model [Royc70] was created to describe the different stages that occur in the development process. To follow the waterfall model, one proceeds from one phase to the next in a purely sequential manner. For example, one first completes "requirements specification" — they set in stone the requirements of the software. When and only when the requirements are fully completed, one proceeds to design. When and only when the design is fully completed, an implementation of that design is made by coders... The waterfall model is thus a top-down approach where the output of each stem serves as input for the next step. This model relies on the controversial hypothesis that these steps can be executed in precise order; there is no jumping back and forth or overlap between steps.

Chapter 4 A support prototype framework for development methodologies

Since this model is very simple, it can easily be used by inexperienced designers as there are not interdependencies between the steps, and no need for a complex coordination between the team members.

However this development cycle presents numerous serious drawbacks that appear as strong limitations to its applicability. First of all many software projects must be open to change due to external factors; indeed clients are notorious for changing their stated requirements while designers are well known for misunderstanding the client's requirements. Whoever should be blamed, the development life cycle must be adaptable since the hypothesis that requirements are stated once for all is clearly utopist.

Secondly, since the people in charge of the different steps of development are likely to differ, it is difficult to Figure out exactly what is needed in each phase of the software process before some time is spent on the next step. Indeed, a feedback from following phases may be helpful to complete previous phases satisfactorily. For example, the design phase may need feedback from the implementation phase to identify problem design areas.

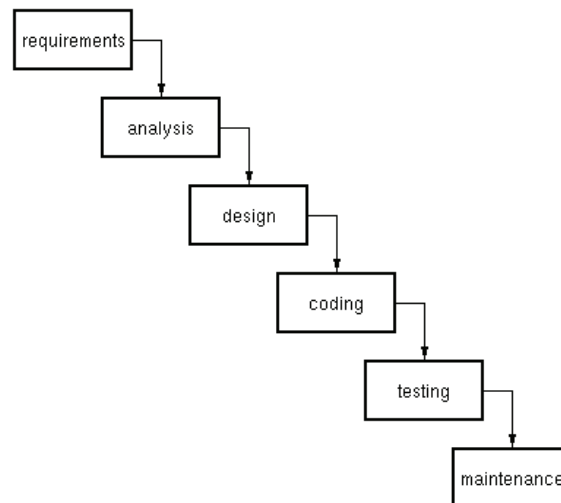


Figure 4-8 Representation of the waterfall development model

This model does not incorporate any risk analysis phase, what could be useful in order to identify potential problems areas of the software development process. This problem is strongly linked with the fact that estimating the time and cost for

Chapter 4 A support prototype framework for development methodologies

each phase of the development process without doing some test in that phase is hard even for a experienced designer.

In response to the problems identified with the genuine waterfall model, many modified waterfall models have been introduced. These models may address some or all of the criticisms of the "pure" waterfall model. One of the alternative version came from Royce himself, his final version intended improvement upon his initial "waterfall model", illustrated that feedback could (should, and often would) lead from code testing to and from design back to requirements specification [Lind01].

With regards to our framework and methodological guidelines, this development life cycle appears to be partially incompatible. Indeed, one the underlying condition to the use of the framework, consist in mixing the activities together. The clear distinction showed between requirements, analysis, and design phase acts as a barrier to the integration of our methodology. As we stated earlier in this chapter, the framework and methodological aspect can be integrated in many existing methodologies as long as these methodologies rely on a frequent interaction with the end users. Fortunately, most of the recent methodologies give a central role to this aspect.

When applied to a waterfall model; the designer has the possibility to use SketchiXML as support for user requirement gathering. In this context, the designer defines all the scenarios from the beginning, as the next development steps only occur once the requirements phase is considered to be competed. Such constraints force the end users to consider all the scenarios at once. So, when this phase is completed, the designer can move to the next step and build the final user interface based on this prototype and build the functionalities extracted from this previous phase. Such prototyping approaches appears thus to be close to a horizontal prototype, even if the end users never see the real user interfaces until the system is fully completed. Of course, such approach presents strong limitation but, they are mainly associated to this development life cycle. By actively involving the end users in the first development steps, we slightly reduce the gap between the users' perception of the systems and the systems itself.

4.2.2 Spiral model

The spiral model [Boeh88] is an iterative development methodology, the basic idea behind this iterative enhancement is to develop a software system

Chapter 4 A support prototype framework for development methodologies

incrementally, allowing the developer to take advantage of what was being learned during the development of earlier, incremental, deliverable versions of the system. The process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving sequence of versions until the full system is implemented. During each cycle, design modifications are made and new functional capabilities are added.

With regards to the waterfall model, the output of each phase is also the input of the next phase, but the iterative aspect offer much more flexibility and enable early detection of errors.

The main purpose of this development model consists thus in minimizing the risk within the development process thanks to an early identification of the potential problem. Moreover, this model is also suited to minimize the resource usage and to ensure well defined quality standards.

The development process is divided into four different activities [Lind01]:

1. *Identify*: This activity consists in identifying the goals, constraints and alternatives to be considered for the respective cycle. The goals define the objectives of the cycle while the constraints specify limit and possibilities on the manner to achieve this goal. Since several paths are likely to lead to the same goals, these alternatives should be elicited.
2. *Evaluate*: based on the alternatives identified previously, the design team evaluates the different alternatives with regards to their respective risk and cost.
3. *Develop*: based on the output of the previous phase, the design team constructs the goal product. To this aim, the alternate process model must be evaluated so as to reflect the quality standard elicited during the first step of the cycle.
4. *Plan*: During this last step of the cycle, the complete cycle is evaluated and the weaknesses are identified. Based on this evaluation, the products and resource for the next cycle are defined. Ideally, all the members of the team should be involved in this phase in order to clarify uncertainties or misunderstandings.

Chapter 4 A support prototype framework for development methodologies

The key concept of the spiral model is thus its capability of early detection error, and pruning amongst the different alternatives. As a result, the spiral model is a very flexible process applicable for a large range of software development project. Moreover, the incremental planning process permit to adjust project plan to changing requirements easier than the other development process model.

The other side of the coin is the need of high management effort to conduct the whole process correctly. Moreover this development process is strongly linked with the Unified Process (UP), thus a commercial product mainly based on the Unified Modeling Language (UML). It appears that this process does not offer the possibility to provide formalization and was not significantly update since the end of nineties.

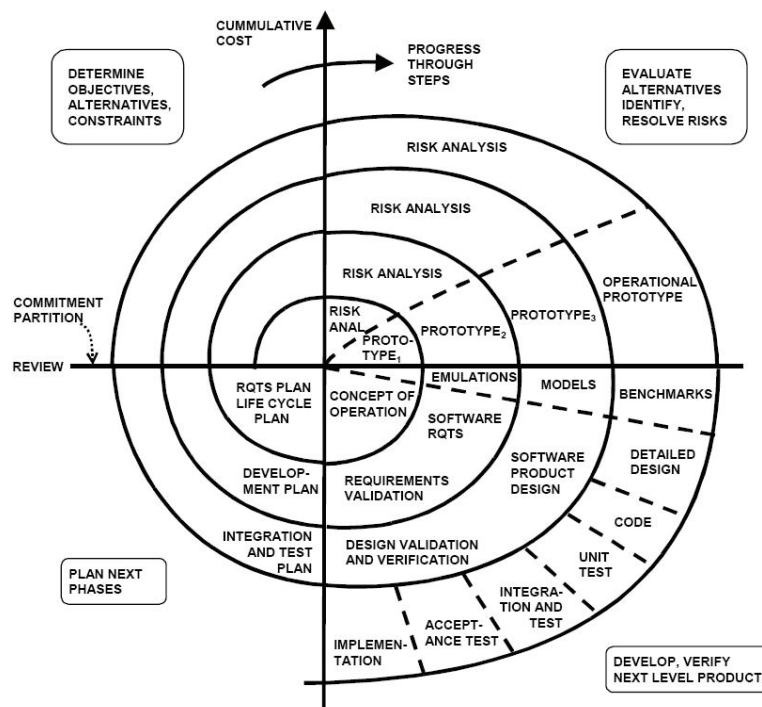


Figure 4-9 The spiral development model [Boeh88]

When considering the spiral model, we observe that the end-users are mainly involved in the requirements elicitation phase, but not really in the design phase; their role in the development process is thus minor. This appears to be relevant since most of the end-users cannot express themselves about the global

Chapter 4 A support prototype framework for development methodologies

architecture. But as we stated previously, the future end-users have at least a graphical idea of what is desired; most of the end-users can express what they need on each user interface and what kind of data should be accessed using this user interface.

As a matter of fact, if we build the mock-up using high fidelity editor, a lot of time shall be wasted on unimportant details, and we will reduce drastically the number of alternative designs explored. Moreover, given that most of the end users have little or none experience in user interface design it is important to have a tool supporting effectively the lack of prerequisites of the end user, and the need of fast iteration. With regard to these requirements SketchiXML appears to provide a valuable answer as it just require the end-user to be able to sketch the desired user interface, without any programming concept. The purpose of the horizontal prototyping step is thus to build together with the end user a prototype of the user interface associated to the requirements elicited in the two other simultaneous steps. The role of the end user is thus to participate to the process and make sure the result match his expectations.

During the two first phases (identify and evaluate), the designer tries to identify the constraints and alternative for the current cycle. Ideally, the designer should choose some objective and elaborate a set of scenario to reach this objective. All the alternatives should be elicited in order to find the most suitable choice. Based on this set of scenario, the end user participate to the evaluation process, as the designer tries to capture all the relevant information associated to this scenario. As a communication means, the designer and the end user use SketchiXML as this tool appears as the most adequate for brainstorming and communicate with the variety of profiles that take place during the development of the application. Once this phase is completed, the designer should have gathered a prototype describing the set of user interfaces that should be part of the scenario, on one hand. On the other hand, the designer should have listed a list of requirements that cannot be expressed on the graphical representation. Based on this information, the designer is able to develop a functional prototype by reusing the output of the previous phases. In addition to the development of the user interface, the designer develops the set of functionalities associated to the scenario being developed. Such process corresponds to a diagonal prototype presented in the second chapter, a functional prototype is being developed with an alternation between user interface design and functionalities development.

Chapter 4 A support prototype framework for development methodologies

During this last step of the cycle, the complete cycle is evaluated and the weaknesses are identified. Based on this evaluation, the products and resource for the next cycle are defined. Ideally, all the members of the team and end users should be involved in this phase in order to clarify uncertainties or misunderstandings.

4.2.3 Extreme Programming (XP)

Extreme programming [Beck04] is a software engineering methodology for the development of software projects based on a grouping of several best practices under a common umbrella. It prescribes a set of day-to-day practices for developers and managers; the practices are meant to embody and encourage particular values.

The best practices considered in XP have 12 practices, grouped into four areas, derived from the best practices of software engineering:

- Fine scale feedback
 - Pair Programming
 - Planning Game
 - Test Driven Development
 - On-site customer
- Continuous process
 - Continuous Integration
 - Design Improvement
 - Small Releases
- Shared understanding
 - Coding Standard
 - Collective Code Ownership
 - Simple Design
 - System Metaphor
- Programmer welfare
 - Sustainable Pace

The goal is to give all developers a common view of the system which matches the view held by the users of the system. To this end, XP favors simple designs, common metaphors, collaboration of users and programmers, frequent verbal communication, and feedback. The on-site customer practice hold a major role is the development process. Oppositely to the classical approach where the software is usually not built for the people that order it, but rather for people who have to

Chapter 4 A support prototype framework for development methodologies

cope with it. Therefore XP require that at least at least one end user must be strongly involved, and be part of the development team. This is particularly important as XP is based on a dynamic approach towards the systems requirements and planning of next steps. Thus the end user part of the team is responsible to provide the required information and feedback.

The basic assumption of the XP appears to be contradictory to the traditional approach, indeed XP assumes that the cost of changes does not raise exponentially as time passes. Oppositely to the more conventional system development methods is the focus on designing and coding for the needs of today instead of those of tomorrow, next week, or next month.

XP supporters acknowledge the disadvantage that this can sometimes entail more effort tomorrow to change the system; but they pretend this is more than compensated for by the advantage of not investing in possible future requirements that might change before they become relevant. Coding and designing for uncertain future requirements implies the risk of spending resources on something that might not be needed.

This model is based on a set of four key concepts that can be found in every phases of the complete method [Lind01]:

- *Communication.* Programmers do not necessarily know anything about the business side of the system under development. The function of the system is determined by the business side, thus the understanding of the required functionalities of the system lie in the communication with the business side.
Moreover, the people within the team must communicate their knowledge to the benefit of the entire group.
- *Simplicity.* XP treats every problem as if it can be solved "extremely simply". Traditional system development methods say to plan for the future and to code for reusability. Extreme programming rejects these ideas.
- *Feedback.* Feedback is a key principle and is most useful if it is done rapidly. The time between an action and its feedback is critical to learning and making changes. In XP, unlike traditional system development methods, contact with the customer occurs frequently in

Chapter 4 A support prototype framework for development methodologies

small iterations. The customer must really be involved and has clear insight into the system that is being developed. He is then able to give feedback and steer the development as needed. So XP appears to be based on the evolutionary prototyping model depicted before, the starting point is always a simple solution that is then enhanced to better ones, thanks to numerous small iteration.

- *Courage.* This means that all the decision are not easy to take in this development process. Indeed, the designers may have to choose between alternative without knowing in advance the potential problems. Secondly, the designers must accept to throw away bad or unnecessary portion of code.

Obviously, Extreme Programming is very controversial. When formal software development processes require change requests to be analyzed and approved by a change control board, in Extreme Programming, the on-site customer makes changes informally, often by verbally informing the development team. Proponents of Extreme Programming claim this makes the process flexible, and saves the cost of formal overhead and critics of Extreme Programming claim this can lead to costly rework and project scope creep.

Other controversial aspects of Extreme Programming are based of the fact that requirements are defined incrementally, rather than trying to get them all in advance. As a matter of fact there is not a big picture of the future system; most of the design activity takes place on the fly from a very simple version adding complexity only when required by failing test. According to critics this could lead the design team to a waste of resource when redesign is needed.

But this methodology has proved in the past to be efficient, and not only for small project. It has been claimed that XP has been used successfully on teams of over a hundred developers. [Lind01].

Nevertheless, Extreme Programming appears to be the most prominent of several agile [High01] software development methodologies. Agile methodologies rank adaptability higher than predictability: the adaptability, to changing requirements, ranks higher than the project predictability valued by more traditional methodologies.

Given the description of the extreme programming, the methodology based on the UsiXML tools appears straightforward. One of the key elements in the

Chapter 4 A support prototype framework for development methodologies

extreme programming and other agile methodologies lies in a strong involvement of the end users during the development. The design process is based on simple design, test driven, small releases..., as many elements that are compatible with the approach advocated in the previous sections of this chapter. But even if this methodology does rely on simple method and strong interaction with the end user, there are no means to ease this communication. Indeed, the main principle of this methodology is to code the system directly without developing any specification or prototype, as such steps are considered as waste of time. As the output of the prototype process can be reused without any effort loss, we recommend to the designers developing application with extreme programming to use the low fidelity prototype tool with their end users. Indeed, SketchiXML match the requirements elicited upper as it permits to involve the end users, it relies on simple concept easy to understand and most of all facilitate the communication between the stakeholders. Even for a skilled programmer, building a user interface is everything but a straightforward process, and most of them tend to sketch the user interface, so as to figure out what kind of constraints should be applied between the components. Building a user interface based on grid bag layout or a spring layout without a paper based support is almost impossible. For a simple user interface with a dozen of widgets, the number of constraints can be as high as one hundred. So, even if the approach recommended by extreme programming consist in coding the application directly, recourse to low fidelity prototype appears very valuable as the result is reused.

As it was the case with the spiral model, extreme programming progresses by small steps so as to validate the development progressively. Thus, based on the interface developed and the constraints gathered the designer develop the logical layer behind these interfaces. This lead us to the same conclusion, we develop a functional prototype with a diagonal approach. Such approach seems to match the expectation of most of the actual methodologies as it cope with most of the problems met during a development process: the mismatch between the user expectation and the designer understanding.

4.3 Conclusion

Through this chapter we have introduced the prototyping framework based on the UsiXML mark up language. This framework composed of three tools, permits to develop prototype with different level of fidelities and maintain coherence between the tools thanks to the UsiXML language.

Chapter 4 A support prototype framework for development methodologies

Based on this prototyping framework, we illustrate how the current methodologies can be improved thanks to a better use of the prototyping techniques. A list of best practice for prototyping with the framework is proposed, on one hand. On the other hand we illustrate how to integrate the framework in some of the existing methodologies.

The next chapter presents three case studies addressing different contexts of use for the framework presented in this chapter.

Chapter 5 Surveys

In order to build and test our application, we carried out a set of surveys at different stages of the development. This chapter presents the four studies that were conducted for this research:

1. The first survey details how the widgets grammar was elaborated.
2. The second survey was conducted shortly after the grammar construction, and consist of a first evaluation of the application. Usability and technical performances are both examined and serve as historical benchmarks for future releases.
3. The third survey evaluates the impact of the fidelity level on the UIs sketching, and collects user preferences.
4. The last survey is aimed at evaluating several aspects. On one hand we captured a lot of information of the performance of the application and simultaneously we observed the complexity associated to the widget and the types of representations.

Following the presentations of the surveys conducted during this thesis, we illustrate the approach developed on SketchiXML with a set of case studies.

5.1 Building a widget catalogue

Once the grammar and interpretation mechanism were completed, the next step consisted is filling the grammar with a set of representation for each widget. For

Chapter 5 Surveys

this purpose, we conducted an experimental study aimed at collecting information on the way users would intuitively sketch the widgets (*R9 - Ease of use*).

5.1.1 Participants

Two groups of 30 subjects were randomly selected from a list of candidates: the first group was composed of people with relevant experience in computer science and interface design, while the second was composed of end users without any specific knowledge in UI design or computer science. The second group was also considered because SketchiXML's goal is to involve as much as possible the end user in the early prototyping process in order to bridge the gap between what they say and what the designer understands. Thus, the representations may vary depending on designers or end users. Figure 5-1 depicts the various domains of expertise of both groups.

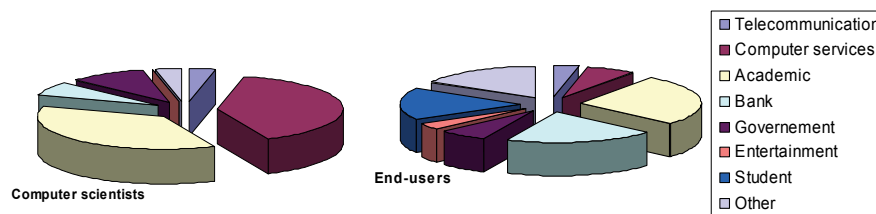


Figure 5-1 Distribution of the subjects according to their domain of expertise.

5.1.2 Methodology

A two phases analysis was carried out on both groups. The scope of the first part was to determine how members of each group would intuitively and freely sketch the widgets to be handled by SketchiXML. From a cross-platform comparison of widgets, a widget catalogue was identified comprising the following 32 widgets:

| | | |
|--------------|------------|---------------|
| text | image | tabbed dialog |
| text field | multimedia | box |
| text area | area | menu |
| push button | layer | color picker |
| search field | group box | file picker |
| login logout | table | date picker |
| reset form | separator | hour picker |
| validate | frame | toggle button |

Chapter 5 Surveys

| | | |
|--------------|-----------|--------------|
| radio button | hyperlink | slider |
| check box | anchor | progress bar |
| combo box | list box | spinner |

Each widget was documented with its English and French name, a screen shot and a small textual description (Table 5-1). For each widget, subjects were asked if they had ever seen this widget before and to provide a sketching representation.



| Widget | Graphical presentation | Textual description |
|-------------------|---|---|
| Search Field |  | This widget is composed of a text field and a button. It allows the users to submit a search. |
| Tabbed Dialog Box |  | This widget allows the user to switch from one pane to another thanks to the tab. |

Table 5-1 Extract of the list of widgets submitted to the participants

Then, from the widget representations provided during the first phase, we tried in a second phase, to extract the most common representations. We grouped all these representations in categories with strong similarities. Table 5-2 presents the most frequent representation provided for a subset of widgets.

From that categorization, we proceeded to the evaluation of each sketch according to a set of criteria:

- *Naturalness*: evaluates if the representation of the widget associated is natural or not.
- *Number of shapes*: considers the number of different of different vectorial shapes in the sketch alternative.
- *Kind of shapes*: evaluates the different kind of shapes involved in the representation such as rectangle, diamond, ellipse, handwriting, gesture...
- *Level of confusion*: estimates if the representation is likely to be confused with the other representations.




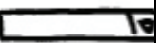





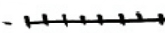
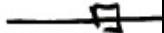



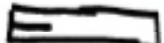



| Widget name | Representations | | |
|-------------|---|--|---|
| Check Box |  |  |  |
| Combobox |  |  |  |
| Listbox |  |  |  |
| Slider |  |  |  |
| Textfield |  |  |  |
| Txtarea |  |  |  |

Table 5-2 Extract of the most common representations provided by the participants

For instance, the first representation of the combo box presents good characteristic as it is only made up of basic vectorial shapes and a pretty low number of shapes. The naturalness aspect seems to be sufficient too, as many participants drew this representation intuitively. The second and third representations were presented less often, and required a large set of shapes and constraints. The same analysis was then conducted for each widget, and leads us to extract a list of representations that could easily be handled by SketchiXML.

Then, we conducted a second survey based on the results of the first one. We built a set presentation proposals based on a combination of the representation provided by the first phase of the survey and a number of new representations.

Chapter 5 Surveys

Participants were then asked to rank the different representations according to their representativeness and preferences as a five point Likert scale. On basis of these results we defined all the representation to be handled by SketchiXML.

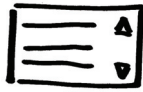




| Representation 1 | Representation 2 | Representation 3 | Representation 4 | Representation 5 |
|---|---|---|--|---|
|  |  |  |  |  |

Table 5-3 Example of the list of representations submitted to the participants for the second part of the survey

5.1.3 Results

Based on the result distribution shown in Figure 5-4, we established the best representation with the following method. Firstly we assessed whether any dependence exists between the participants. If this first step's results established a significant dependence, we then proceeded to the second phase and we computed the aggregate preference of both groups and the global preference. For each widget, the Kendall coefficient of concordance W [Sieg88] test was computed. This coefficient expresses the degree of association among n variables, that is, the association between n sets of rankings. The degree of agreement among the 60 people who evaluated the representations is reflected by the degree of variation among the 6 sums of ranks.

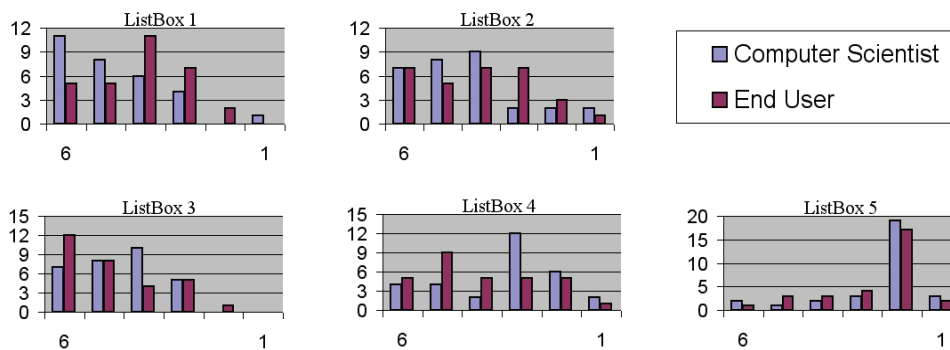


Figure 5-4 Result frequency for each representation proposed in table 5-3

$$W = \frac{\sum_{i=1}^N (\bar{R}_i - \bar{R})^2}{N(N^2 - 1)/12} = 0,36238$$

Figure 5-5 Computation of W where k is the number of judges, N the number of objects being ranked, \bar{R}_i the average of the ranks assigned to the ith object, \bar{R} the average of the rank assigned across all objects or subjects and $N(N^2-1)/12$ represents the maximum possible sum of the squared deviations

The comparison of the value obtained from this computation to the critical value shows that the null hypothesis (independence between participants) has to be rejected. We can thus proceed to the second phase of the analysis and establish a ranking among all representations using the Borda Count method [Bord81]. The principle of the Borda Count Method is that, each candidate gets 1 point for each last-place vote received, 2 points for every next-to-last-place vote, etc., all the way up to N points for each first-place vote where N is the number of candidates. On the basis of this analysis we observed that both groups had almost the same preferences among the representations (Figure 5-4). Most of the time, the set of well considered representation is the same even if small changes in the sequence occur. Out of this results set, we considered the preferred representations with respect to their intrinsic complexity as explained earlier. For instance, list box 4 obtained a good score compared to the other representations, but its intrinsic complexity is very high as it requires hand writing recognition, that was not supported at the moment. List boxes 4 and 5 were thus discarded from the final selection. Often, the set of representations selected for the list box is composed of the three first representations depicted in the corresponding set of representations.

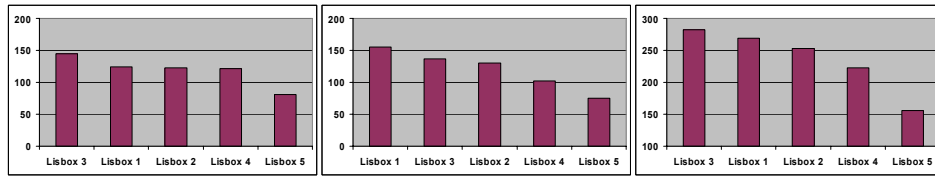


Figure 5-6 Borda Count results for end users, computer scientists and both categories aggregated

Thus, the representations to be considered for the list box are three first representations proposed on Table 5-3. The method was then extended to all the widgets in order to build a catalogue of widgets' representation. The current widget catalogue proposes sketching alternatives for all the graphical components

that can be found in UsiXML. All the representations can be found in Appendix C.

5.2 Testing the application

According to the ISO-9241 norm (www.usability.net), the usability defines the effectiveness, the efficiency and the satisfaction with which users achieve a specific goal in a particular environment. The usability test refers thus to a process that employs representative participants of a target population to evaluate to what degree a product meet specific usability criteria.

So, in order to evaluate the usability of the application, we conducted a large-scale survey on 40 participants. The objective of this test is thus to validate that the usability objective, such as the speed, accuracy, ease of use, naturalness, defined in (*R9 – Ease of Use*) are met. A secondary objective of this test was the creation of a historical record of usability and performances benchmarks for future releases.

5.2.1 Participants

Two groups of 20 subjects were selected in a list of candidates: the first group was composed of people with relevant experience in computer science and interface design, while the second was composed of end users without any specific knowledge in UI design or computer science. As it was the case with the first survey, we also considered a group of end-user since SketchiXML's goal is to involve as much as possible the end user in the early prototyping process. Figure 5-7 depicts the domains of expertise of both groups.

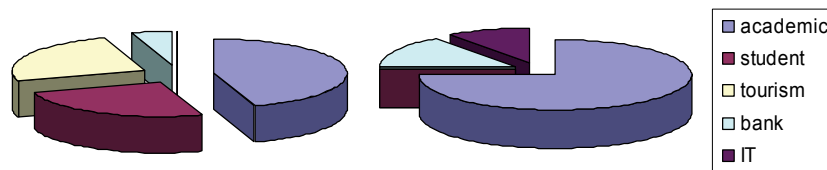


Figure 5-7 Distribution of the subjects according to their domain of expertise

5.2.2 Methodology

Each participant was asked to draw three forms after a five minutes training with the tool. The first and third forms were similar as they were used to evaluate if the

Chapter 5 Surveys

time needed to draw a given form decreased as the participants use the application. While participants were sketching the user interfaces (forms), the test supervisor collected all the information related to the performances such as the number of shapes recognized correctly, the number of unrecognized strokes.

The screenshot shows a web browser window with a registration form titled "Formulaire d'inscription". The form is in French and includes the following sections:

- Renseignements personnels**: Fields for Nom, Prénom, and Date de naissance.
- Coordonnées permanentes**: Fields for Adresse, Ville, and Région.
- Formation**: Fields for État d'avancement, Spécialisation, Diplôme, and Précisions.

At the bottom of the form, there is a checkbox labeled "J'autorise Place aux jeunes à utiliser ces renseignements ou à les transmettre à quiconque est en mesure de me soutenir dans mon cheminement de carrière." and a "Valider" button.

Below the form, there is a question: "Par quelles instances ou quels moyens as-tu entendu parler du site Acro des régions?" with three checkboxes: "Place aux jeunes", "les établissements d'enseignement", and "les transports en commun".

Figure 5-8 First form submitted to the participants of the survey

The analysis is divided in two sections. The first section considers the behavior in terms of performance while the second section focuses on the usability aspects.

a. Performance

As stated in the state of the art, an informal design tool supporting sketch recognition must support it almost perfectly (*R4 – Robust Recognition*). Indeed, if the designer has to redraw a shape several times before the tool finally recognizes it, then the advantages of such tools might become doubtful.

For each widget drawn we collected all the information associated to the performance:

- *Number of widgets correctly recognized*: evaluates the number of widget for which all the shapes part of the representation were correctly recognized.

Chapter 5 Surveys

- *Number of shape correctly recognized*: evaluates the number of shape correctly recognized (even if the widget was not recognized)
- *Number of non-recognized shape*: consider the number of shapes that were not recognized at all.

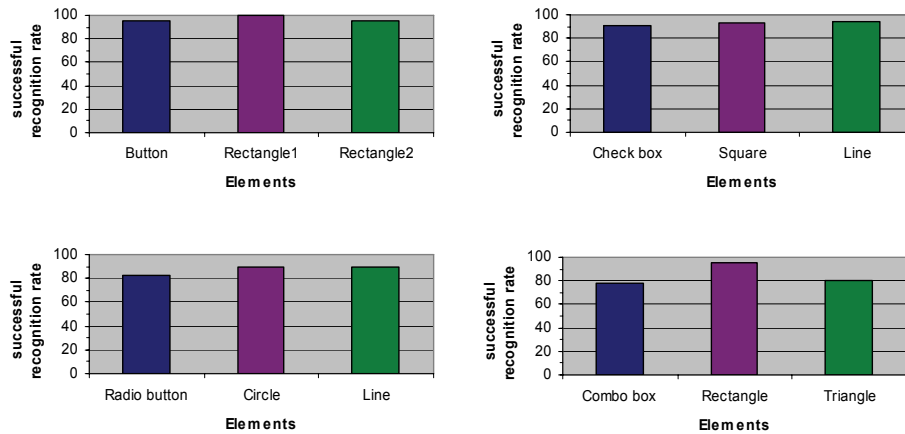


Figure 5-9 Recognition rate for the button, check box, radio button and combo box widget, with the recognition rate of each vectorial shapes part of the widget

Figure 5-9 illustrates the successful recognition frequency for the check boxes, radio buttons, buttons and combo boxes.

| Shape | General rate |
|-----------|--------------|
| circle | 92,85% |
| triangle | 89,99% |
| text | 93% |
| square | 93,4% |
| rectangle | 94,14% |

Table 5-3 Distribution of the subjects according to their domain of expertise with a weighted average of 93,01%

Table 5-3 gives a summary of the average correct recognition rate for each kind of vectorial shapes present in both forms. As it appears clearly on the chart, some of the vectorial shapes show an average recognition rate. Indeed, as it was stated in the requirement, such application should support recognition perfectly, however we are far below the 95 % expected. For instance the average recognition rate for the triangle is close to 90%. But, if we only consider the recognition rate for the

Chapter 5 Surveys

triangle of the combo box, then it falls to 80 %. On the other hand, the successful recognition rate for the triangle of an image component is close to 100%. The weighted average recognition rate for the shapes is 93.01 %.

Table 5-4 shows the recognition rate for each widget present in both forms. As the recognition rate of the vectorial shape was quite low, the recognition rate for the widget is thus very low too. If we consider the weighted average of the recognition rate for the widgets, then it falls far below 90% with 86.88 %. It comes out that the tool should be drastically improved as requirement (*R4 – Robust Recognition*) is not met.

| Widget | General Rate |
|--------------|--------------|
| Text Field | 84,67% |
| Button | 95% |
| Radio button | 83,10% |
| Check box | 90,2% |
| Image | 93% |
| Combo box | 78% |
| Text | 94,12% |

Table 5-4 Distribution of the subjects according to their domain of expertise with a weighted average of 86,88%

Nevertheless, some interesting observation emerged from these tests. It seems that the recognition rate would be correlated with the size of the shape drawn. Indeed, this would make sense as noises can represent a huge proportion of the dots set when the stroke is small. In order to test this assumption we conducted a two-sample test for binomial proportions between the triangle drawn for the combo box (small sketch) and the image component (large sketch). The test compares p_1 , the proportion triangle recognized for the image with p_2 , the proportion of triangle recognized for the combo box. Then the target parameter about which we will test a hypothesis is $(p_1 - p_2) > 0$. In this case, the p value computed is 3.06 and falls outside of the acceptance and leads us to conclude that the proportion of small triangles recognized correctly is smaller than for the large triangles.

So as to solve this problem, we have introduced a zooming function to the application. Indeed, at the time we conducted the survey, some of the functionalities presented in Chapter 3 were not integrated in the application.

Chapter 5 Surveys

The type of input device is also likely to explain the misrecognition as participants were asked to draw the user interface on a graphic tablet “Wacom Intuos²” (Figure 5-10). Unfortunately, the drawing area of such tablet does not contain any reference marks such as a grid. It is thus very difficult to draw on the tablet without looking at the screen and vice-versa. Many participants pointed this problem as the major drawback of the application.

In order to verify this assumption, the application should be tested on different types of device in order to evaluate the impact of the device used.



Figure 5-10 The graphic tablet Wacom Intuos²

We also tested if the recognition rate was better as participant got more used to the application. To this end, we computed the Cramer's V [Sieg88], a statistical measuring the strength of association or dependency between two (nominal) categorical variables in a contingency table.

We also tried to see if the successful recognition rate was function of the background of the participants in terms of prototyping experience. But, here again we did not find any relevant difference.

As a conclusion for this section, it appears that the backgrounds of the participants do not influence the recognition rate of the application, but the performances do not really meet the expectations desired. Indeed the average recognition rate is lower than 90%, and is thus far too low. Further development of the application should then focus on that matter.

Several solution paths should be considered:

- It appears the size of the drawing area might have a positive effect. Thus, a new set of tests should be conducted for the next versions so as to evaluate the impact of that factor.

Chapter 5 Surveys

- The use of the zoom should also be tested. Indeed, several people tried the new version with the zoom feature and reported that it was easier to sketch small shapes than with the previous one, unfortunately we cannot conclude anything.
- A last path to consider, closely related to the first one, is the impact of the input device used. Indeed, many participants reported in the usability questionnaire, problems to draw on the graphic tablet while watching the screen and the lack of visual reference marks on the graphic tablet.

b. Usability

Through this section we evaluate how people interacted with the application. Did they like the application? Was it easy to use? Did it meet their expectations?... are all questions that will be addressed in this section.

The first part of this section aims at evaluating the learning effect and the importance the participants' background when using the SketchiXML. To this end we analyzed the time needed to draw the different forms with regards to their background as a designer and the previous experience with the application.

Figure 5-11 shows the time distribution to draw the first form. It is distributed on an interval ranging from 100 seconds to 385 with a standard deviation of 73 seconds and an average of 187 seconds.

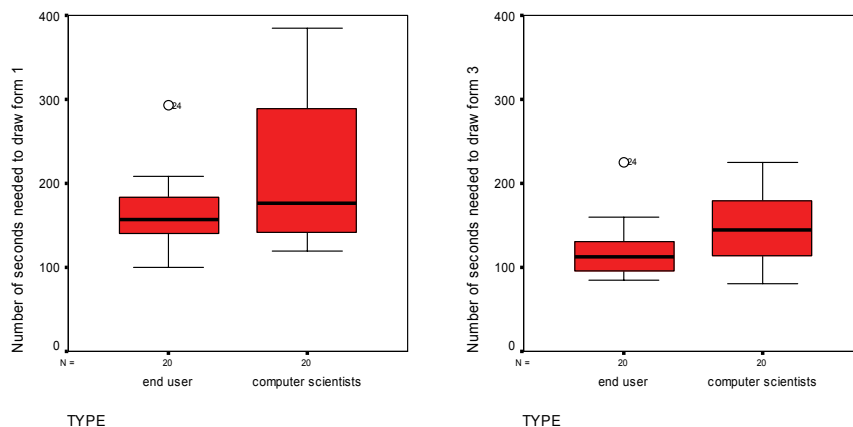


Figure 5-11 Time needed to complete the test for both group of participants

Chapter 5 Surveys

In order to test if the time needed to draw the first form decrease as designer uses the application we conducted an ANOVA test on the average time needed to draw the form. Thus, the first step consisted in testing the homogeneity of variances of the samples. Indeed, the analysis of variance assumes that variances are equal across groups or samples. To this end, we computed the Levene test [Sieg88]. As the homoscedasticity assumption is met, the results of the analysis of variance test is valid.

The outcome of the test showed that the time needed to draw the third interface is significantly lower than the time needed to draw the first one for both groups. As explained in the previous section, the error rate was the same between the first and the third forms, we can thus conclude that a learning effect exists as the participants need less time for the same result after only a minimal training of the application. (*R10 – Ease of Use*)

We conducted the same test to see the relationship between the time needed and the type of the participant. The results are very surprising; indeed, the time needed to draw the first form for the computer scientist is significantly different than the time needed for the end-users. End-users showed better performance than the computer scientists.

The next step was conducted in order to identify a possible link between the age of the participants and the time needed to draw the form. To this aim, we computed the Pearson correlation coefficient [Sieg88]. The result showed that there is no correlation the time needed to draw the first form and the age of the respondents.

The last part of this section present some interesting results among all the questions asked to the participants at the end of the experiment. Trough these results, it appears that most of the participants enjoyed the use of the application. Indeed, even if it does not appears on the charts presented hereunder, most of the participants expressed very positive impression on their experience with tool. It was the first experience with a graphic tablet for most of them, and they really seemed to like the concept. Moreover, they found that the application was easy to use and were quite satisfied with the performances.

Chapter 5 Surveys

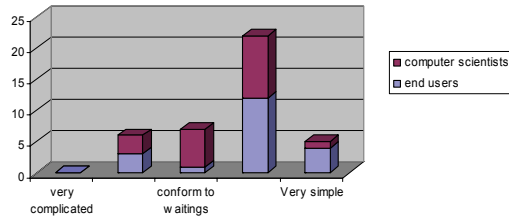


Figure 5-12 Question: How would you categorize the tool's usage?

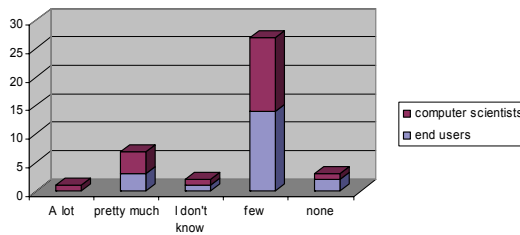


Figure 5-13 Question: Have you faced many problems when using the tool?

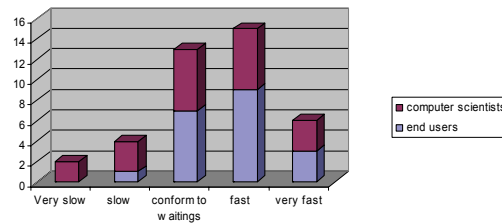


Figure 5-14 Question: what do you think of the speed of the application?

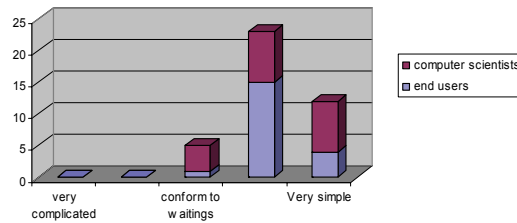


Figure 5-15 Question: How would you consider apprenticeship to the application?

As a conclusion, it's clear that the requirements in terms of performances were not met. But on the other hand, most of the participants were really satisfied with the performance provided by the tool. Indeed, in addition to the multiple choice

questionnaire submitted at the end of the test, participants had the opportunity to write comments on the application. Most of these comments were positive and just a minority of the participants evoked the poor recognition rate as a problem. Moreover, the fact that the time needed to draw the first and the third form is different is also encouraging as it means that learning effect is pretty strong.

Thanks to this survey, we have proceeded to the creation of a historical record of usability benchmarks for future releases. It emerges thus that future development of the application should focus on the performance improvement and on the impact of the input device used with the application.

5.3 Experimental Study on Fidelity Levels

In order to evaluate how end users and UI designers appreciate the various fidelity levels at design time, an experimental study has been set up for investigating the effects of fidelity level on a UI design activity by sketching.

5.3.1 Participants

Twelve volunteers participated in this study. Participants ranged in age from 23 to 39 years ($M=30$ years), including 6 females and 6 males to keep gender balance. Participants were selected on the basis of general inclusion criteria including age and profile (end user or UI designer). All participants were identified and recruited regarding their job in the computer science area (e.g., regular users, computer science researchers, developers, and UI designers from private companies). Table 5-5 summarizes the demographic information and the characteristics of the overall participant sample. Age represents the average number of years for the overall sample. Gender represents the frequency counts of males and females. General profile denotes the frequency in categories: end users vs. UI designers. Professional computer experience represents the average number of years for the overall sample in designing computer experience represents the average number of years for the UI designers only. The end users versus designers assessment was made in order to obtain a comprehensive profile of participants.

| N | Age | Gender | | Handedness | General profile | | Computer experience | |
|----|-----|--------|--------|------------|-----------------|--------------------------|---------------------|----------------|
| | | Male | Female | | End users | User interface designers | Professional exp. | Designing exp. |
| 12 | 30 | 6 | 6 | Right | 6 | 6 | 5.25 | 4 |

Table 5-5 Summary of participants' demographics and characteristics

Table 5-5 summarizes the demographic and the characteristics of the participants based on the grouping. Age represents the average number of years for each

Chapter 5 Surveys

participant group. Gender represents the frequency counts of males (M) and females (F) within each group. Professional computer experience represents the average number of years for each participant group. Designing computer experience represents the average number of years for the designers only.

| Group | N | Age | Gender | Professional exp/ | Designing exp. |
|----------|---|-----|----------|-------------------|----------------|
| Designer | 6 | 31 | M=4, F=2 | 6.8 | 4 |
| User | 6 | 29 | M=2, F=4 | 3.7 | N/A |

Table 5-6 Summary of group profiles

5.3.2 Apparatus and experimental task environment

The computer system used in this study was a PC Dell Latitude D820 equipped with an Intel Core 2 Duo T7200 (2.0 GHz, 4 Mo cache level 2 memory) processor and 2 Gb of RAM memory. Participants were seated approximately 30 cm from a 21-inch Wacom Cintiq 21UX touch screen flat panel connected to this computer. Screen resolution was set to 1,600 x 1,200 pixels, with a 32-bit color palette. The keyboard was not required to complete the task since the participants were supposed to use a stylus. The sketching tool used in this experiment is the one whose implementation has been described in next subsection. The experimental task to be carried out by participants consists of designing two UIs (combined in a pair) in each of the following fidelity levels: Lo-Fi, Me-Fi, Hi-Fi, or No-Fi. Each UI contains eight widgets amongst the following alternatives: push button, check box, combo box, list box, progression bar, radio button, spinner, text area, and text field. A UI pair is considered to be complete once the eight widgets of both UIs have been entirely designed with the imposed fidelity level.

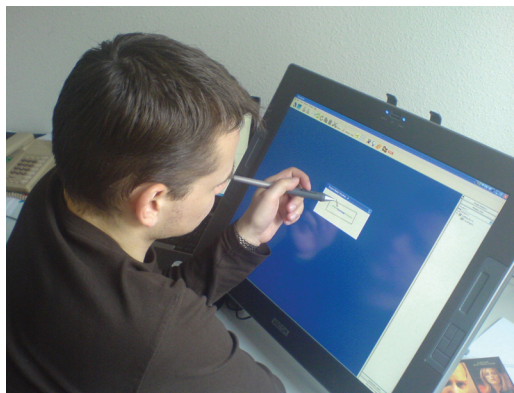


Figure 5-15 A participant sketching on the Wacom Cintiq

5.3.3 Methodology

Prior to experiment, participants were given an explanation of the research study and their role in the study. Following completion of the demographic questionnaire, the participants were briefed on how to use the setup and how to carry out the task. A short training period has been allocated for each participant to sketch a given UI until they feel confident in using the setup. They were also allowed switching between the four fidelity levels. The main part of the experiment consisted in designing four pairs of windows in a pre-assigned fidelity level. The order of the four pairs of windows was randomly assigned. After the sketching tasks, participants were asked to complete a Computer System Usability Questionnaire (CSUQ, see Appendix E) [Lew95] and were interviewed according to a semi-structured scheme. The interview focused on their subjective satisfaction and perception about the study, the system and their preferences in term of fidelity level. The dependent variable used to assess participant task performance was Window Development Time (WDT), which represents the task duration (in seconds) required by a participant to design a window.

5.3.4 Results

The following section is divided into 3 parts: the first subsection explains the statistical outcome of the analysis, while section 2 and 3 present the qualitative result of the survey.

a. Statistical analysis

One participant has not followed the instruction related to the order of the conditions. Consequently, the sample includes only 88 entries instead of 96. Due to the sample size, an analysis of variance (ANOVA) was used to examine the presence of significant differences in task performance, as measured by WDT. Table 3 reproduces the results of two analyses: influence of the fidelity level and influence of the user profile. The statistical significance is underlined.

| ANOVA | Tests of Sig. Diff. Between groups |
|---------------------------------|------------------------------------|
| 1) Fidelity (No/Lo/Me/Hi-Fi)- | F=1.8888; p=0.1377 |
| 2) User profile (User/Designer) | F=7.2719; p=0.0084 |

Table 5-7 Tests for significant differences in performance

Although results from Table 5-7 show that the fidelity level had no influence on WDT, Hi-Fi demonstrated the fastest WDT (M= 261 seconds), respectively followed by No-Fi (M= 297 seconds), Me-Fi (M= 359 seconds) and Lo-Fi (M= 376 seconds) (Figure 5-16 a). In addition, the results from Table 5-7 show that

Chapter 5 Surveys

user profile had a significant influence on WDT ($F=7.2719$; $p=0.0084$). Unsurprisingly, participants from the end users group demonstrated the fastest WDT compared to those from the designers group (respectively, $M=267$ seconds versus $M=369$ seconds - Figure 5-16 b). Indeed, this observation had already been done in previous surveys.

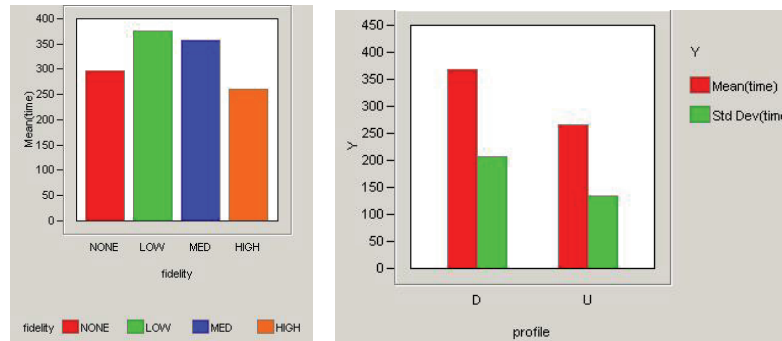


Figure 5-16 (a & b) Mean WDT (seconds) for each fidelity condition, Mean WDT (seconds) for each participant group

b. Computer System Usability Questionnaire.

The IBM CSUQ [Lew95] is a public domain instrument to measure user satisfaction with computer system usability in the context of scenario based usability studies. The CSUQ is made up of four parts, each consisting of items ranked on a 7-point Likert scale: the overall satisfaction score (OVERALL: all 18 Items), the system usefulness score (SYSUSE: Items 1-8), the information quality score (INFOQUAL: Items 9-15), and the interface quality score (INTERQUAL: Items 16-18). This questionnaire has been chosen because of its acceptable reliability: a coefficient alpha exceeding .89 for all parts has been proved. Seven-point rating scales (1=totally disagree, 7=totally agree) were used because they allow three levels of either positive or negative ratings.

| Subscale | Statistical Indices | | |
|-----------|---------------------|--------|---------------|
| | Mean | Median | Std deviation |
| SYSUSE | 4.04 | 4 | 1.52 |
| INTERQUAL | 5.39 | 6 | 1.14 |
| OVERALL | 4.83 | 5 | 1.17 |
| INFOQUAL | 4.45 | 4.5 | 1.37 |

Table 5-8 Summary of overall sample CSUQ. Statistical indices are mean, median and standard deviation.

Table 5-8 suggests that the system usefulness is moderately appreciated as well as the information quality (reasonably good mean, but large deviation). However, the interface quality and the overall user satisfaction are both assessed positively.

c. Subjective general comments and users preferences.

One third of the participants judged the stylus uncomfortable because of a physical button located too close to their index finger. One third of the participants reported that some system functionalities were not usable: the copy-paste was estimated too slow and required too many pointing gestures; the lack of drag-and-drop of sketched items was regretted since it is at the present time replaced by the cut-paste functionality. One third of the participants considered that the speed of the recognition should be improved in the next version of the tool. In return, three quarters of the participants judged the tool as user-friendly and intuitive. This result is consistent with the INTERQUAL result reported above (Table 5-8). Moreover, eight on 12 participants considered the tool as fast and accurate in term of drawings/sketchings recognition. Finally, most of the participants reported a pronounced preference for Hi-Fi (5 participants on 12, including 2 designers and 3 users) and Me-Fi (5 participants on 12, including 3 designers and 2 users). They argued they felt “more comfortable” in those two levels because of the real-time interpretation of their drawings and the resulting UI aesthetics. Furthermore, 75% of the participants dislike the No-Fi (9 participants on 12, including 4 designers and 5 users). They claimed that this level “looks like a draft”, which is consistent with [Meye05].

5.3.5 Interpretation and discussion

The experimental task used in this study was a simplified version of a UI development life cycle. Time required by participants to develop UIs (WDTs) was used as an indicator on the usability of the fidelity levels. This metric revealed its shortcoming: WDT is not exact enough to be considered as representative of participant performance. Further usability studies need to include other metrics like the number of recognized/unrecognized shapes/texts/gestures, as well as the number of effective “widgets” that are added to the interface.

The statistical analysis revealed no significant impact of the “fidelity level” parameter on the user performances (speed). This result may be due to the fact that the level of fidelity has no influence on the sketching strategies adopted by the users, that is to say they perform the tasks in the same way, no matter what

the level of fidelity is. In addition, the statistical analysis revealed a significant impact of the user profile (end user vs. designer) on the performances. Surprisingly, end users –with no experiment in interface design– are faster in performing sketching tasks than designers. This result may be due to the fact that designers do care a lot about the quality and aesthetics of the resulting interfaces (e.g., they systematically preserved alignment, symmetry, and semantic grouping of UI elements) compared to end users. Consequently, more time is required for designers to sketch valuable interfaces, regarding their own personal criteria. These results are consistent with some earlier findings [Coye05].

Finally, the qualitative analysis revealed a pronounced user preference for both Hi-Fi and Me-Fi. This result suggests that participants, including both end users and designers, may prefer in terms of visual comfort, visual feedback, and widget recognition the fidelity levels that show a resemblance to the final UI. Differences observed between end users and designers are consistent with some other findings [Bail03 ,Coye05, Virz96, Walk02].

5.4 Evaluating the representations

The first versions of SketchiXML were only supporting one level of fidelity and were using a set of representations defined according to an experimental study. Even if we were convinced that the number of stroke or the type of constraint held a central role, we never addressed this aspect rigorously. Analogously, we have always considered that the level of fidelity of the interpretation should be minimal. Thought the next section we propose an experimental study aimed at evaluating the complexity associated to the widget supported by SketchiXML and the impact of the fidelity level. This process is divided in a series of steps. First, we detail the widgets that are taken into account for the test with a visual representation and a set of information related to the relation to be checked amongst their atomic components. Second, based on the set of widgets presented in the first sub-section, we present into detail the purpose of the evaluation study. Third, the methodology to be used in the experimental is described. Fourth, the result obtained are presented and commented into details.

5.4.1 Widgets Taxonomy: an a priori classification

The sketching activity in SketchiXML is considered as sketching combinations of shapes and constraints. Indeed, as shown in Table 5-9, each widget is detailed as a precise combination of atomic components. Based on this set of components, a

Chapter 5 Surveys

graphical code is defined as a set of constraints that must be satisfied. Table 5-9 provides an illustration of the graphical code associated to each widget.












| Widget | Gesture representation | # | Atomic components | Graphical code | | | | | | |
|---------------|---|---|-------------------------------------|----------------------|------------------|-------------------|---------------|--------------|------------------------|------|
| | | | | Specific orientation | Simple inclusion | Complex inclusion | Juxtaposition | Intersection | Sequence of components | Size |
| Button |  | 2 | Rectangle (2) | | | X | | | | |
| Checkbox |  | 2 | Rectangle + Line | X | | | X | | | X |
| Combobox |  | 2 | Rectangle + Line/Triangle | | | X | | | | X |
| Label |  | 1 | Line | X | | | | | | |
| List box |  | 5 | Rectangle + Triangle (2) + Line (2) | X | | X | | | X | X |
| Picture |  | 2 | Circle + Triangle | | | | | X | | |
| Progress bar |  | 3 | Rectangle | X | | X | | | X | X |
| Radio button |  | 2 | Circle + Line | X | | | X | | | X |
| Slider |  | 2 | Line + Triangle | X | | | | X | | X |
| Text area |  | 3 | Rectangle + Line (2) | X | X | | | | X | |
| Text field |  | 2 | Rectangle + Line | X | X | | | | | |
| Toggle button |  | 3 | Rectangle (2) + Line | X | | X | | | | |

Table 5-9 Widgets Taxonomy

As we observe on table 1, the visual code refers to a visual “semantic” that is based on 7 criteria:

- *Specific orientation* is a unary constraint that evaluate if the atomic component has a precise orientation. Refer to the check box, the label, or the progress bar to illustrate the horizontal orientation code; refer the toggle button to illustrate the vertical orientation code.
- *Simple inclusion* is binary constraint applied between two components that evaluate whether an atomic component is inside another or not. (See text field and the text area)

- *Complex inclusion* is similar to the simple inclusion with the exception that additional constraints are specified. This constraint is considered as more complex as it may require an atomic component to be enclosed in a particular region of another component. As a consequence, the atomic component to be enclosed is likely to be very small, and thus increase the difficulty. (see the button, the combo-box, the list box, the progress bar and the toggle button.
- *Juxtaposition* is a binary constraint that requires two atomic components to be in juxtaposition. (see the radio button and the checkbox)
- *Intersection* is a binary constraint that requires two atomic components to intersect. (see picture and slider)
- *Sequence* is a constraint that implies a repetition of a same atomic component. (see list box, progress bar and text area)
- *Size* is a constraint that implies a specific size for an atomic component (small, large...)

The visual semantic of the widgets' representation was built upon the visual grammar of diagram elements (node-link diagrams) in [Ware,2004].

From this observation, it appears natural to consider that the complexity associated to each widget will not only be dependent on the number of its atomic components, but also dependent of its visual code. For instance, sketching a progress bar may be more difficult than sketching a text area in spite of a same number of atomic components (both the progress bar and the text area count three atomic components). Similarly, sketching a list box or a toggle button is likely more difficult than sketching a label.

The presentation of the widgets in Table 5-9 was build with respect to the widgets' name (alphabetic order). No a priori assumption about the eventual difference between widgets' complexity was formulated at this stage. The potential influence of widgets' representation is studied in the experimental usability study presented in the following.

5.4.2 Current study objectives

As stated previously, the purpose of the experimental study presented here consists in addressing one of the major issues related to user interfaces prototyping, the potential influence of widgets'representations in a low fidelity prototyping tool. To achieve this goal we asked a group of users to sketch a set of common widgets, such as labels, combo-boxes, sliders, progress bars, etc. Table 5-

9 presents the components used in the test and their respective visual semantic. The main purpose of this experimental study is thus to evaluate the impact of the representation associated to each widget, but also to evaluate the impact of the fidelity level on the sketching process.

5.4.3 Methodology

This section is divided into three subsections, first we proceed to a description of the test procedure, then the participants profile are detailed and we conclude with the presentation of the experimental design. The hardware setup to be used in this experiment is exactly the same than the previous survey (7.3).

a. Procedure

Each participant received a detailed explanation of the research study. Following the short introduction to the test procedure and test purpose, all participants performed some training with the tool. Following the training session, participants performed the series of widget sketches with a constant randomization of both the widget to be sketched and the fidelity level to be used. Simultaneously, all the relevant data were stored in a log file to be used for statistical analysis.

b. Profile

Eleven volunteers participated to this experimental study, five females and six males. This group of participants was composed of experienced computer users, aged between 22 to 28 years. Moreover, all the participants were considered as expert pen users, as they had significant past experiments with on pen-based interaction.

c. Experimental design

The survey was based on a 4x12x2 factorial design; 4 fidelity levels were evaluated (none, low, medium and high), 12 widgets (see Table 1) and each widget was repeated twice for each fidelity level. So, all participants received exactly the same 96 triplets (fidelity, widget, repetition) to sketch. However, the presentation sequence of these 96 triplets was randomized so as to neutralize a potential task learning effects. The dependent variables used to assess the participant task performances were the widget sketching time ST (from the first stroke to

recognize to the effective recognition of the widget), and the accuracy (i.e., number of delete operations DEL and number of superfluous stroke operations SST to sketch a given widget).

The main directive for the participants was to sketch the component chosen by the application, as fast as they could. Participants were asked by the system to sketch a given widget at a time. A dialog box was asking them to sketch a specific widget in a specific fidelity level. In particular, neither the fidelity level nor the widget could be changed by the user. Once the user considered the widget to be finished he had to click on one of the lateral button of the screen to move to the next component. If the widget to draw was present on the drawing surface, the surface was cleared and a new widget was proposed to the participant, else the participant was asked to finish the current component.

5.4.4 Results

In order to identify the potential impact of the different element introduced in the previous section, analyses of variance (ANOVA) were conducted. Thanks to these analyses, we have examined the presence of significant differences in task performance as measured by widget sketching times (ST), the number of delete operations (DEL) and the number of superfluous stroke operations (SST). These tests were carried out with regards to:

- the fidelity level
- the subject ID
- the widget type.

If significant differences were revealed by the ANOVA procedure, complementary analyses, including mean comparisons, classifications and partitionning, were also performed to highlight differences. These quantitative results are presented in the following.

a. ANOVA procedure

The results of the ANOVA procedure are presented in Table 5-10. Factors are fidelity level, widget type and subject ID. Variables are widget sketching times (ST) in milliseconds, number of delete operations (DEL) and number of superfluous stroke operations (SST). Highly significant influences are underlined.

| Subject ID | DF | ST (ms) | DEL | SST |
|----------------|----|-----------------------|----------------------|----------------------|
| Fidelity level | 3 | F=2.1039 p=0.0981 | F=2.5627 p=0.0535 | F=1.9866 p=0.1143 |
| Subject | 10 | F=6.1802 p<0.0001 | F=5.5658 p<0.0001 | F=2.6989 p=0.0029 |
| Widget type | 11 | F=11.5065 p<0.0001 | F=2.7674 p=0.0022 | F=7.3756 p<0.0001 |

Table 5-10 ANOVA Procedure

The results from Table 5-10 show that contrary to what was expected, fidelity level is not a significant factor: it has influence neither on widget sketching times nor on the number of superfluous stroke operations; however, we observe a tendency for the number of delete operations ($F=2.5627$, $p=0.0535$). This result is in accordance with earlier findings [Coye07] and suggests that fidelity level is transparent for single widget sketching tasks. In addition, the results from Table 5-10 suggest that both widget type and subject ID are significant factors. Indeed, the type of widget and the time needed for its construction are clearly linked. We also observe that some widgets are likely to involve a higher error rate as the number of superfluous strokes and delete operations are dependant of the widget type. We can observe the same trends for the subjects. Consequently, the complementary analysis, performed in order to elicit the specific influence of widget complexity on subjects' performances, needs to take into account not only the widget type variable, but also the subject id variable. To achieve this goal, a hierarchical classification of the subjects was carried out beforehand the specific analysis of the widget complexity per group of subjects. Results are presented in the following.

b. Subject classification

We estimated that the method of Student's t-test was not relevant enough for performing the complementary analyses, because, mainly, of the illegibility of the results. Thus, instead of t-tests, we used graphical representation of the subjects' performances such as a plot diagram of sketching time according to mean and median and the recursive partitioning technique. Recursive partitioning (RP) was applied to the dataset in order to elucidate statistically significant sub-groupings within the data by relating subjects' performances (ST) to the subject id factor. The resulting graph and decision tree are presented respectively in Figures 5-17 and 5-18.

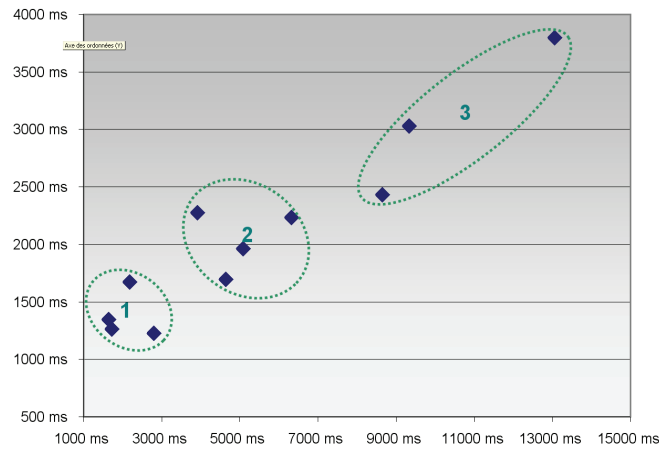


Figure 5-17 Subjects' performances (ST) per mean and median

The results from Figures 5-17 and 5-18 show that there are three different groups of participants:

- G1: participants # 3, 5, 6, 11 (M=2092 ms, Std Dev=2920);
- G2: participants # 1, 7, 9, 10 (M=4995 ms, Std Dev=9868);
- G3: participants # 2, 4, 8 (M=10347 ms, Std Dev=24802).

According to both the average sketching time (M) and its associated standard deviation (Std Dev), G3's performances have been removed from the data set. Indeed, the slow sketching time (M=10347 ms) associated to a very high standard deviation (Std Dev=24802) justifies this choice, especially by considering these bad performances in comparison with those observed for G1 and G2. The average sketching time observed for G3 is five times the one observed for G1 and twice the one observed for G2.

Based on this finding we have had a closer look to the log file generated by the application; we observed that for two of the three participants present in the group G3, SketchiXML had bugged. Even if we are not certain that the application misbehaves for the third participant, they can be considered as outliers and removed from the data set for the remaining of the analysis. Thus, the following of the analysis focus only on the two selected groups (G1 and G2).

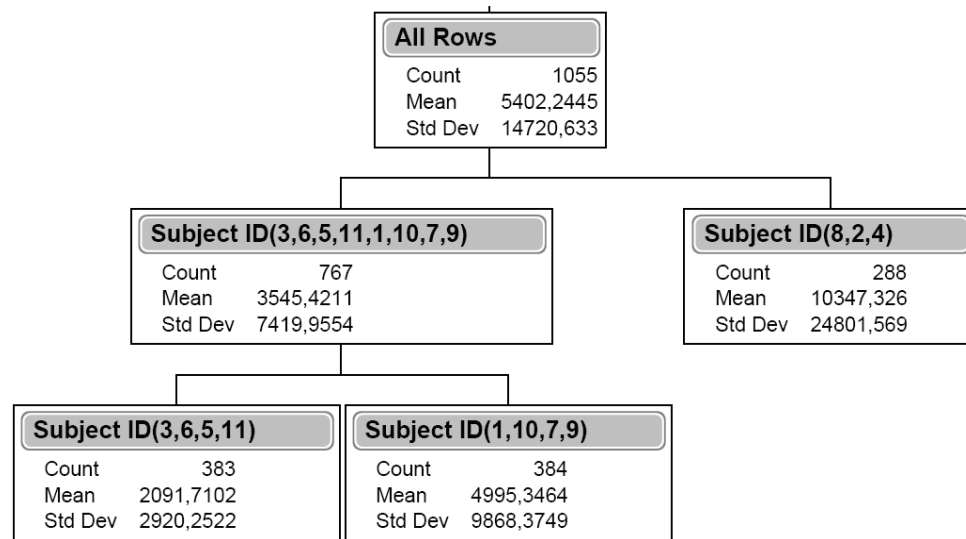


Figure 5-18 Participants classification

c. Widget classification

The results per widget presented in table 5-11 are related to G1 and G2 groups and include:

- recognition rate in percent,
- sketching times in milliseconds (ST), number of superfluous strokes (SST) and delete operations (DEL) considering both mean (in ms) and standard deviation.

Widgets are sorted by increasing average sketching times (ST). To elicit differences between groups of widgets, a complementary Student's t-test was performed on data and it has highlight some differences between three groups of widgets:

1. tooglebutton, listbox and progressbar,
2. button, textarea, combobox, slider, picture, radiobutton, checkbox, textfield,
3. slider, picture, radiobutton, checkbox, textfield, label.

Chapter 5 Surveys

| Group | Widget | Recogn. rate (%) | Sketching Time | | Superfluous stroke | | Delete operations | |
|-------|--------------|---------------------|----------------|---------|--------------------|---------|-------------------|---------|
| | | | M (ms) | Std Dev | M (ms) | Std Dev | M (ms) | Std Dev |
| 1 | textfield | 0.98 | 1688 | 4528 | 0,047 | 0,375 | 0,031 | 0,250 |
| 1 | label | 0.97 | 734 | 4973 | 0,078 | 0,513 | 0,047 | 0,278 |
| 1 | picture | 0.97 | 1913 | 2254 | 0,095 | 0,560 | 0,079 | 0,451 |
| 2 | button | 0.95 | 3322 | 6595 | 0,156 | 0,71- | 0,078 | 0,370 |
| 1 | checkbox | 0.95 | 1694 | 1643 | 0,172 | 0,579 | 0,078 | 0,410 |
| 1 | radiobutton | 0.94 | 1764 | 2719 | 0,187 | 0,639 | 0,109 | 0,441 |
| 2 | textarea | 0.94 | 3259 | 3574 | 0,250 | 0,992 | 0,094 | 0,387 |
| 2 | slider | 0.92 | 2607 | 3841 | 0,266 | 0,980 | 0,156 | 0,597 |
| 3 | listbox | 0.91 | 7241 | 11271 | 0,937 | 2,525 | 0,437 | 1,446 |
| 2 | combobox | 0.86 | 3246 | 4663 | 0,406 | 0,955 | 0,297 | 0,885 |
| 3 | progressbar | 0.84 | 6448 | 8452 | 0,672 | 1,584 | 0,328 | 0,944 |
| 4 | togglebutton | 0.80 | 8603 | 15729 | 1,031 | 2,462 | 0,562 | 1,542 |

Table 5-11 Results per widget (ST, SST, DEL)

Due to the lack of accuracy of this result, i.e., in order to get more accurate differences between groups of widgets, we applied a recursive partitioning (RP) to the dataset. The principle of RP is to elucidate statistically significant subgroupings within the data by relating subjects' performances (ST) to the widget type factor. The result of such process gives the decision tree presented Figure 5-19. It shows that there are four different groups of widgets:

- W1: Label, textfield, checkbox, radio button, picture;
- W2: Slider, combo box, textarea, button;
- W3: Progressbar, listbox;
- W4: Togglebutton.

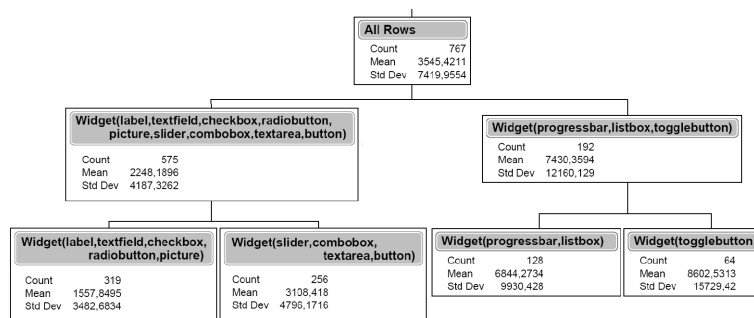


Figure 5-19 Widgets classification

This finding is consistent with what was expected: according to widget sketching time, the complexity of a widget (expressed for instance in terms of specific

Chapter 5 Surveys

orientation, inclusion, or size) has a significant impact on user performances, such as speed (ST), as well as accuracy (SST, DEL and recognition rate). According to the RP decision tree (see Figure 5-19), sketching a toggle button or a list box is likely to be more difficult than sketching a label. Furthermore, we observe that representation including a larger number of constraints (specific orientation only for the label in comparison with specific orientation + complex inclusion + sequence of components + size for the list box) tends to be associated to a highest error rate. In the following section, we present the detailed analysis of the influence of factors such as specific orientation, complex inclusion, sequence of components, or size of components on user performances.

These results partially validate our taxonomy of widget's representation and complexity (see table 5-12). First, they suggest that both the criterion of complex inclusion ($F=15.0896$, $p=0.0001$) and the concept of specific orientation ($F=13.2787$, $p=0.0003$) do have a significant impact on user performances while widget sketching speed. Secondly, one observes a tendency according to which the criterion of juxtaposition would have an influence on this same performance.

| Factor | DF | F Ratio | Prob>F |
|------------------------|----|-------------|------------|
| Specific orientation | 1 | $F=13.2787$ | $p=0.0003$ |
| Simple inclusion | 1 | $F=0.3673$ | $p=0.5446$ |
| Complex inclusion | 1 | $F=15.0896$ | $p=0.0001$ |
| Juxtaposition | 1 | $F=2.2567$ | $p=0.1333$ |
| Intersection | 1 | $F=1.1249$ | $p=0.2891$ |
| Sequence of components | 1 | $F=1.6698$ | $p=0.1966$ |
| Size | 1 | $F=0.8742$ | $p=0.3500$ |

Table 5-12 Anova Procedure with ST(ms) as variable

d. Learning effects

An analysis of variance (ANOVA) was computed to compare users performances by quartile; the analysis shown that the quartile is a significant factor considering the sketching times ($F=2.7795$, $p=0.0402$). In addition, quartile is a significant factor neither for the superfluous stroke operation ($p>0.05$), nor for the delete operations ($p>0.05$).

Consequently, a complementary analysis was performed on user performances par widget and per quartile. The results are presented in Table 5-13 according to both the recognition rate and the sketching time per quartile (columns Q1, Q2, Q3 and Q4) and for the overall session (columns “total”).

| Widget | Recognition rate | | | | | Sketching Time | | | | |
|------------------|------------------|------|------|------|-------|----------------|---------|---------|---------|---------|
| | Q1 | Q2 | Q3 | Q4 | total | Q1 | Q2 | Q3 | Q4 | total |
| button | 0.88 | 0.94 | 1.00 | 1.00 | 0.95 | 6654 | 2646 | 1952 | 2036 | 3322 |
| checkbox | 1.00 | 0.94 | 0.88 | 1.00 | 0.95 | 1221 | 2040 | 2030 | 1486 | 1694 |
| combox | 0.75 | 0.88 | 0.94 | 0.88 | 0.86 | 4501 | 2445 | 2859 | 3178 | 3246 |
| label | 0.94 | 0.94 | 1.00 | 1.00 | 0.97 | 2513 | 353 | 35 | 37 | 734 |
| listbox | 0.88 | 0.94 | 0.88 | 0.94 | 0.91 | 7325 | 7250 | 8535 | 5852 | 7241 |
| picture | 1.00 | 1.00 | 1.00 | 0.87 | 0.97 | 1858 | 1541 | 1378 | 2836 | 1903 |
| progress | 0.69 | 0.81 | 0.94 | 0.94 | 0.84 | 11429 | 5697 | 4118 | 4547 | 6448 |
| radio | 1.00 | 0.94 | 0.88 | 0.94 | 0.94 | 1085 | 2030 | 1913 | 2029 | 1764 |
| slider | 0.94 | 0.88 | 0.94 | 0.94 | 0.92 | 3313 | 2789 | 2474 | 1851 | 2607 |
| textarea | 0.94 | 1.00 | 0.81 | 1.00 | 0.94 | 3545 | 2342 | 4838 | 2312 | 3259 |
| textfield | 0.94 | 1.00 | 1.00 | 1.00 | 0.98 | 3379 | 1080 | 1089 | 1204 | 1688 |
| toggle | 0.69 | 0.69 | 0.88 | 0.94 | 0.80 | 16041 | 6641 | 7445 | 4283 | 8603 |
| total | 0.89 | 0.91 | 0.93 | 0.95 | 0.92 | 5238,62 | 3071,19 | 3222,15 | 2637,61 | 3542,39 |

Table 5-13 Statistics per widget

First and foremost, results per quartile suggest that there is a task learning effect: on the one hand, the global recognition rate increases between the first quartile (0.89%) and the last one (0.95%); on the other hand, the average sketching time decreases between the first quartile (M=5239 ms) and the last quartile (M=2638 ms). According to these observations, we can conclude that there is a learning effect of the task.

e. Conclusion

With regards to statistical analyses we have observed some significant results. Firstly we have observed that the level of fidelity did not have any impact on the sketching of an individual widget. Naturally, such observation does not imply that a prototyping tool can choose to use indifferently a level of fidelity of another. Indeed, the fidelity is likely to influence the creation of the complete user interface, as some representation may give an impression of almost finished results, as an example. Here, we only demonstrate that the time needed to build a given widget, is not dependant of the fidelity level to be used.

Oppositely and unsurprisingly, we have also demonstrated that the quality of the recognition was significantly dependant of the type of widget and of the users. Through this section, we can observe that participant can be grouped into three different subgroups according to their respective performances. Even if few information can be extracted from such finding, it was necessary to take this aspect into account for the interpretation of the dataset. The observations made on the widget representation are richer as they provide valuable information for

Chapter 5 Surveys

the development of a new grammar, and for the improvement of some part of the application. We observed strong differences between the widget representations. Based on a recursive partitioning approach we defined a set of group a similar widget based on the time needed for their construction. We have grouped then this information with the average recognition rate associated to the widget in order to refine the grouping.

| Group | Widget | Gesture representation | Recognition rate | Specific orientation | Simple inclusion | Complex inclusion | Juxtaposition | Intersection | sequence of components | Size |
|-------|---------------|---|------------------|----------------------|------------------|-------------------|---------------|--------------|------------------------|------|
| 1 | Text field |  | 0,98 | X | X | | | | | |
| 1 | Label |  | 0,97 | X | | | | | | |
| 1 | Picture |  | 0,97 | | | | | X | | |
| 2 | Button |  | 0,95 | | X | | | | | |
| 2 | Checkbox |  | 0,95 | X | | | X | | | X |
| 2 | Radio button |  | 0,94 | X | | | X | | | X |
| 2 | Text area |  | 0,94 | X | X | | | | X | |
| 2 | Slider |  | 0,92 | X | | | | X | | X |
| 3 | List box |  | 0,91 | X | | X | | | X | X |
| 3 | Combobox |  | 0,86 | | | X | | | | X |
| 3 | Progress bar |  | 0,84 | X | | X | | | X | X |
| 3 | Toggle button |  | 0,80 | X | | X | | | | |

Table 5-13 Widget representation ranked by recognition rate

Table 5-13 illustrates these observations, the last four elements, are the widget that required the more time with the highest error rate. We can observe that all of the four widgets are built using complex inclusion in addition to more simple graphical codes. Moreover, the ranking of the widget illustrates that a larger set of constraints tend to increase the recognition rate and the time required. This lead to us to the following conclusion, when defining representation for widgets **a minimal number of constraint should be used**, especially when ambiguities between the representations are unlikely. For instance, the text field

representation requires the enclosed line to be horizontal, but the line could be drawn with many other orientations for the same results as there are not any other representation composed of a single rectangle and line. Also, the fact that the recognition rate for the toggle button is very low can probably be explained with other reason than the intrinsic complexity of the widget. Indeed, this widget seemed very easy and we expected a recognition rate almost similar to the button. This low recognition rate may result from a constraint that would not be well defined.

The last significant observation made during this survey is related to the learning effect; we observed for all the participants that their overall performances were significantly higher at the end of the survey. They drew the widget more precisely, as the recognition rate is higher, in less time. Obviously, these two observations are linked; the lower time at the beginning of the test can be partially attributed to the numerous delete operations.

5.5 Case studies

In order to validate the use of SketchiXML and its integration with the other UsiXML tools, this chapter presents three case studies. These case studies were selected such as to illustrate different contexts of use with several targets platforms and languages. Indeed, depending of the target user interfaces SketchiXML will be adapted. Thus such variations of contexts permit to illustrate the different mode and development path present in SketchiXML. The first case study is similar to the case study that was used for the state of the art; it consists thus in designing a web site selling several types of media. The second prototype is a movie finder website for pocket pc; this simple web site for pocket pc allows the users to search for a movie with a limited set of criteria. The third case study consists in a java wizard for GrafiXML configuration.

5.5.1 E-media

The E-Media Shop [Faul04a, Coye03] is a store selling and shipping different kinds of media items such as books, audio CDs, videotapes, software and the like. E-Media Shop customers (on site or remote) can use a periodically updated catalogue describing available media items to specify their order. As most of the competitors present on the market, the shop proposes to the client to proceed to a wide set of operation. First of all, the client can browse the catalogue and obtain information about all the products present in the database. Naturally, the client has the possibility to buy the items by adding the desired items to his virtual

Chapter 5 Surveys

shopping cart. Once the customer has finished browsing the catalogue he can proceed to the payment of the items present in the shopping cart. In order to accomplish that, the client must be registered on the website. If this condition is respected, the customer proceeds directly to the payment; otherwise the system required the user to register first. If the user is already a customer, he just needs to fill in login and password, if it is his first purchase on the website, he is asked to fill a form with a set of general information about him.

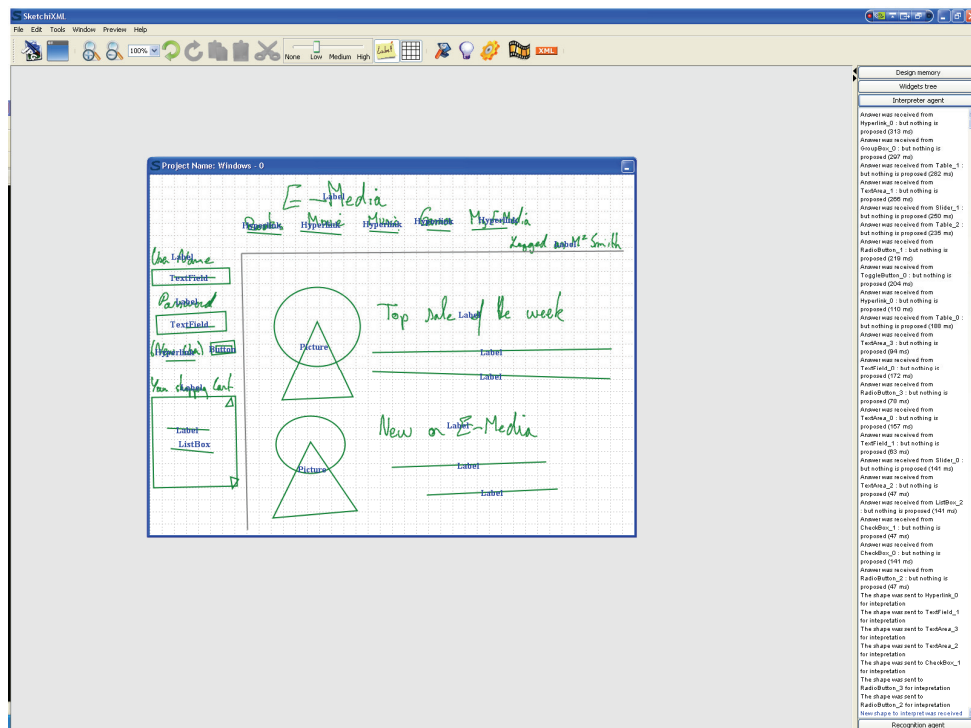


Figure 5-20 SketchiXML main interface, building the e-media home page

Figure 5-20 illustrates the main interface of the web site. This interface contains three main parts, the navigation area on the top of the UI that is a set of hyperlink allowing to navigate between the different sections of the web site. On the left of the web page, we have two main parts. One hand we have a set of component for the login process. As we can find on any web site requesting user identification, we ask for a user name and a password. Below these fields, the user can either click on the button to proceed to login, and if he has not created an account yet, then the “new user” hyperlink permits him to create an account on the web site. On the other hand, we have the shopping cart that is displayed. The shopping cart is made up of a list of all the items that were selected, and a button allowing to see

Chapter 5 Surveys

the content into details. The central part of the web page contains all the information specific to this specific page while the two previous set of components are presents on each page of the web site.

Based on this general layout, we proceed next to the description of the other web pages. Figures 5-21 (a) and 5-21 (b) illustrate such process. We copy all the common part of the main interface and past it on a new window. Then we can proceed to the enriching of the template with the specific information associated to the target page. Figure 5-21 (b) corresponds to the form allowing to the end user to create a new profile on the web site. To this aim he is asked to provide a set of general information about him.

The process is similar for the remaining of the web site, the user copy-pastes the parts of the interface that are constant on each page, and sketches the components that are specific to the new user interface.

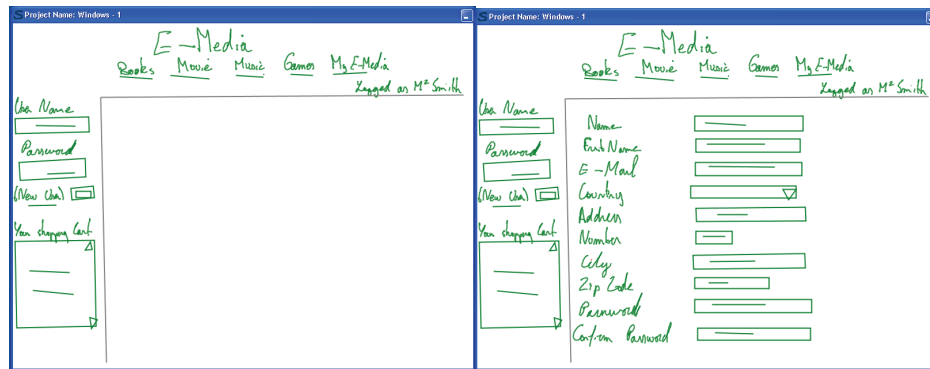


Figure 5-21 (a & b) replicating the common part of the web page thanks to SketchiXML edit functionalities

At any moment, the user has the possibility to switch from the design perspective to the scenario edition perspective. In the example provided in Figure 5-22, we define the global navigation in a first time. On this example, we have designed the main interface and a couple of transition between the main components of the web site. Amongst all the components present in the web site, we only cover a small sub set as the process is repeated for each part of the web site. The example presented in this case study covers the “create account” function, search for an item, display shopping cart, and proceed to payment of the shopping cart.

Chapter 5 Surveys

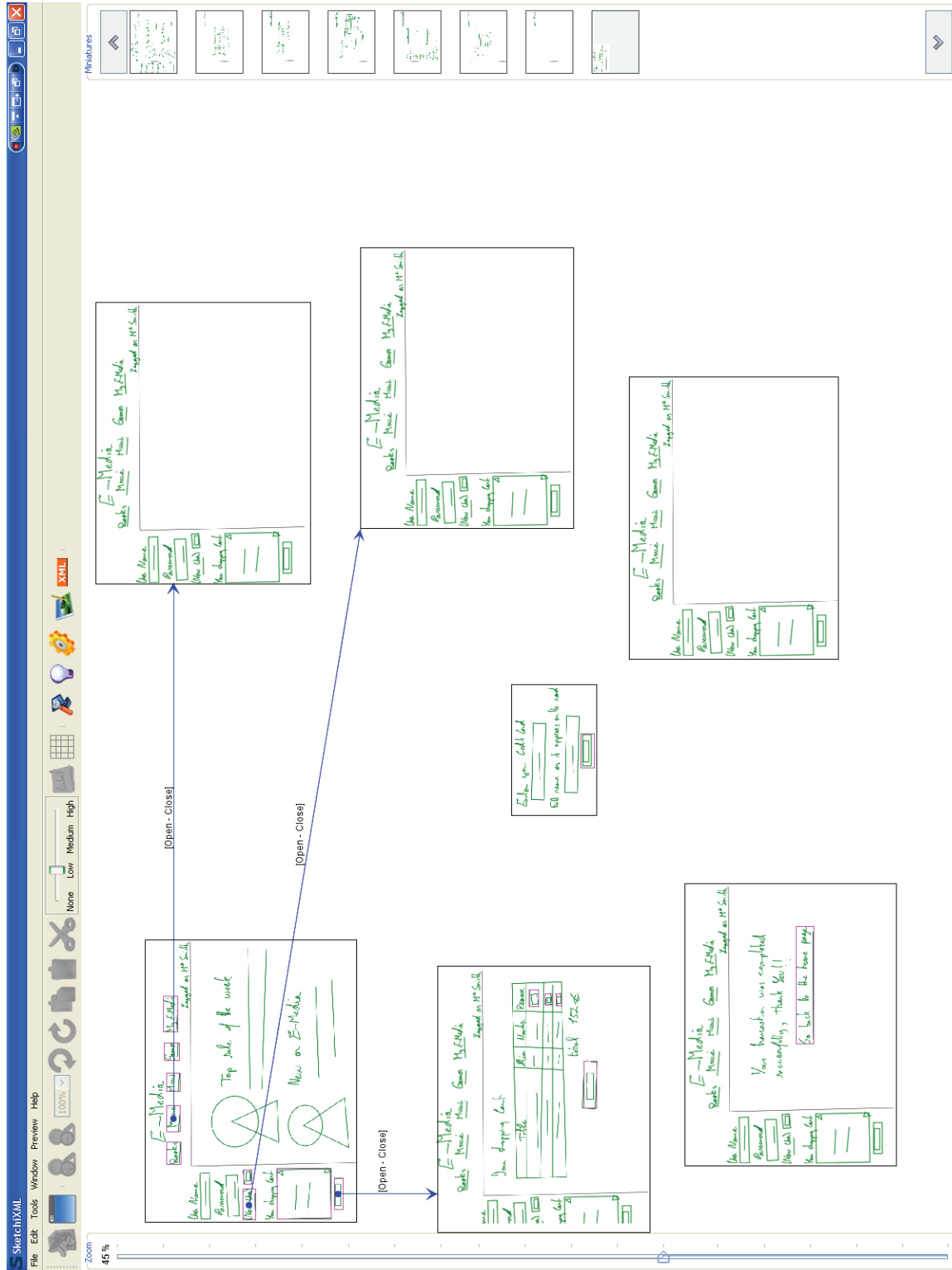


Figure 5-22 Specifying the general structure of the web site with the navigation editor

Chapter 5 Surveys

Obviously, when we will switch to a higher fidelity editor, the common part of the user interface will be refined once, and propagated to all the windows. To this aim there is no need to specify that the target of each link present in the navigation bar for each window. Defining it once is sufficient and permits to have a lighter result; otherwise the screen would be overloaded.

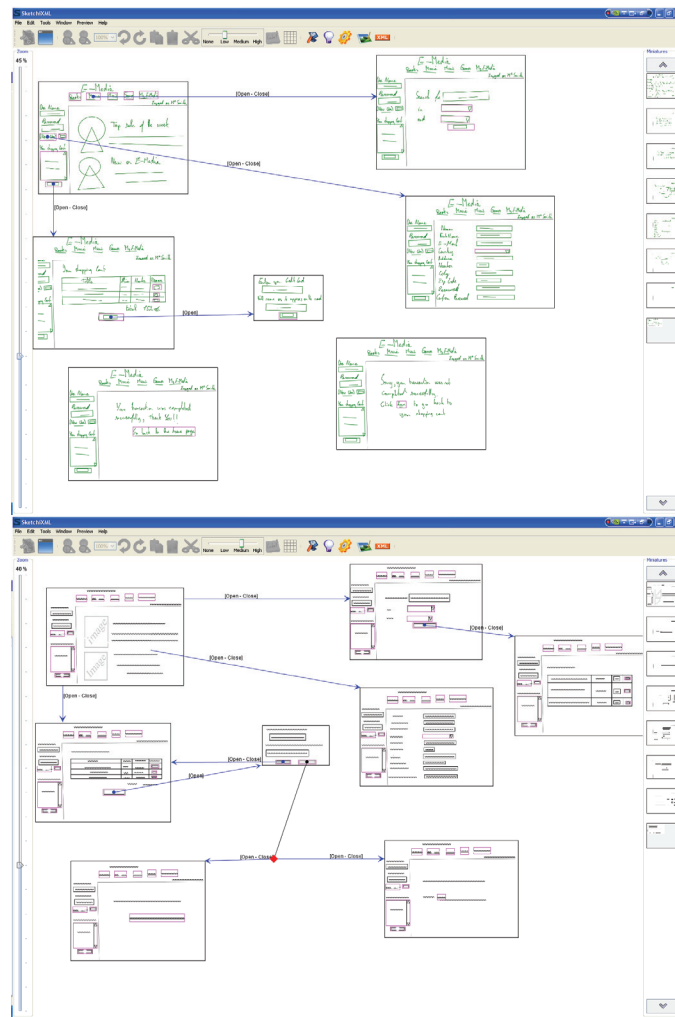


Figure 5-23 (a & b) Based on the general description of the web site, we continue to specify the global navigation of the web site

As we are developing a web site, most of the transitions between the windows are “open-close” transitions, as the navigation on a website is usually moving from one page to another, this seems to be consistent.

Chapter 5 Surveys

Following this short description of the navigation, we keep on refining the project and detail the other interface into details, and we specify the navigational aspect of the application deeper into details.

Figure 5-23 displays a set of UIs completed on basis of the template elaborated in a first time. Based on this set of UIs, we switched to the medium fidelity representation and we specified other relationships between the user interfaces. For instance, when the user click on the validate button associated to the shopping cart, a dialog box is displayed, and ask to the user to fill provide financial data for the transaction. Following the results of this test, a web page is displayed, and informs the customer about the status of the transaction by either displaying an error message or thanking the customer for his purchase. This example is far from being exhaustive; indeed, we could also specify what happen when the client clicks on a item after a query, or the fact that a client has to be logged before proceeding to the purchase of the items in the shopping cart... and many other improvement. As, the design of other components and relations is not different from the example described in Figure 5-23, it would redundant to specify the remaining of the case.

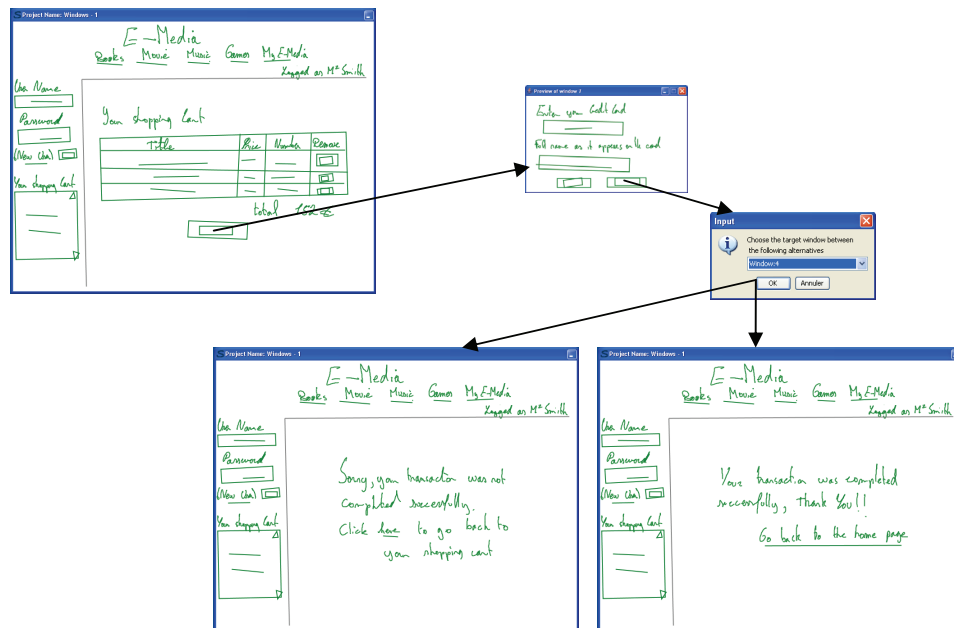


Figure 5-24 SketchiXML run-mode permits to visualize the prototype in action

Chapter 5 Surveys

Based on this sketching phase, we are now able to evaluate the navigation of our prototype. To this aim, we use the overview mode present in SketchiXML, and we keep a low level of fidelity. Indeed, as it was presented in chapter 3, SketchiXML also permits to pre-visualize the system by generating the corresponding UIs in Java. However, the user is free to choose the level of fidelity sketch for the rendering, but also for the navigation editor and the run mode. In this context, visualizing the set of web page using a high fidelity representation based on java toolkit is not very judicious; indeed the look and feel of a web page is completely different than a windows application. For this reason we prefer to stay at a low level of fidelity so as to avoid confusion between the environments.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <uiModel id="EMedia" name="EMedia" creationDate="2007-03-29T17:25:09.468+02:00"
  schemaVersion="1.8" xmlns="http://www.usixml.org">
- <head>
  <version modifDate="2007-03-29T17:25:09.468+02:00" />
  <authorName>Adrian</authorName>
  <comment>This file was generated with SketchiXML</comment>
  <comment>Information on this tool can be found on www.usixml.org</comment>
</head>
- <guiModel id="EMedia-cui" name="EMedia-cui">
- <window id="window_0" name="window_0" isVisible="true" isEnabled="true" width="800"
  height="599" isAlwaysOnTop="false" isResizable="true">
- <gridBagBox id="Box_0" name="Box_0" gridHeight="29" gridWidth="40">
- <constraint gridx="11" gridy="0" gridwidth="8" gridheight="2" weightx="1.0" weighty="1.0"
  fill="none">
  <outputText id="Label_0" name="Label_0" isVisible="true" isEnabled="true"
  fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false"
  isStrikethrough="false" isSubscript="false" isSuperscript="false" isPreformatted="false"
  textColor="#000000" textFont="Dialog" isItalic="false" visitedLinkColor="#000000"
  textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
+ <constraint gridx="8" gridy="3" gridwidth="3" gridheight="2" weightx="1.0" weighty="1.0"
  fill="none">
...
+ <constraint gridx="10" gridy="20" gridwidth="7" gridheight="9" weightx="1.0" weighty="1.0"
  fill="none">
- <constraint gridx="19" gridy="20" gridwidth="13" gridheight="2" weightx="1.0" weighty="1.0"
  fill="none">
  <outputText id="Label_9" name="Label_9" isVisible="true" isEnabled="true"
  fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false"
  isStrikethrough="false" isSubscript="false" isSuperscript="false" isPreformatted="false"
  textColor="#000000" textFont="Dialog" isItalic="false" visitedLinkColor="#000000"
  textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="8" gridy="3" gridwidth="3" gridheight="2" weightx="1.0" weighty="1.0"
  fill="none">
  <outputText id="Hyperlink_0" name="Hyperlink_0" isVisible="true" isEnabled="true"
  fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false"
```

Figure 5-25 UsiXML specification produced by SketchiXML


```

<?xml version="1.0" encoding="UTF-8" ?>
- <uiModel id="Project_Name" name="Project_Name" creationDate="2007-03-
  29T17:25:09.468+02:00" schemaVersion="1.8"
  xmlns="http://www.usixml.org">
+ <head>
- <guiModel id="Project_Name-cui" name="Project_Name-cui">
- <window id="window_0" name="window_0" isVisible="true" isEnabled="true"
  bgColor="#ece9d8" width="921" height="709" isAlwaysOnTop="false"
  isResizable="true">
- <gridBagBox id="Box_0" name="Box_0" gridHeight="35" gridWidth="46">
+ <constraint gridx="7" gridy="3" gridwidth="5" gridheight="2" weightx="1.0"
  weighty="1.0" fill="none">
+ <constraint gridx="13" gridy="3" gridwidth="5" gridheight="2" weightx="1.0"
  weighty="1.0" fill="none">
+ <constraint gridx="19" gridy="3" gridwidth="5" gridheight="2" weightx="1.0"
  weighty="1.0" fill="none">
- <constraint gridx="25" gridy="3" gridwidth="5" gridheight="2" weightx="1.0"
  weighty="1.0" fill="none">
  <outputText id="Hyperlink_3" name="Hyperlink_3"
    content="/uiModel/resourceModel/cioRef[@cioId='Hyperlink_3']/resource/@
    content" defaultContent="Music" isVisible="true" isEnabled="true"
    fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false"
    isStrikethrough="false" isSubscript="false" isSuperscript="false"
    isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
    defaultHyperLinkTarget="http://www.usixml.org" visitedLinkColor="#000000"
    textMargin="0" numberOfColumns="15" numberOfLines="1" />
  </constraint>
+ <constraint gridx="31" gridy="3" gridwidth="5" gridheight="2" weightx="1.0"
  weighty="1.0" fill="none">
+ <constraint gridx="32" gridy="5" gridwidth="9" gridheight="1" weightx="1.0"
  weighty="1.0" fill="none">
- <constraint gridx="1" gridy="8" gridwidth="6" gridheight="1" weightx="1.0"
  weighty="1.0" fill="none">
  <inputText id="TextField_0" name="TextField_0" isVisible="true" isEnabled="true"
    fgColor="#000000" bgColor="#ffffff" textColor="#000000" maxLength="100"
    numberOfColumns="20" numberOfLines="1" isPassword="false"
    isWordWrapped="true" forceWordWrapped="true" isEditable="true" defaultFilter=""
    />
  </constraint>
- <constraint gridx="1" gridy="10" gridwidth="5" gridheight="1" weightx="1.0"
  weighty="1.0" fill="none">
  <outputText id="Label_3" name="Label_3"
    content="/uiModel/resourceModel/cioRef[@cioId='Label_3']/resource/@cont
    ent" defaultContent="Password" isVisible="true" isEnabled="true"
    fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false"
    isStrikethrough="false" isSubscript="false" isSuperscript="false"
    isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
    visitedLinkColor="#000000" textMargin="0" numberOfColumns="15"
    numberOfLines="1" />
  </constraint>
.....
  <outputText id="Label_4" name="Label_4"
    content="/uiModel/resourceModel/cioRef[@cioId='Label_4']/resource/@cont

```

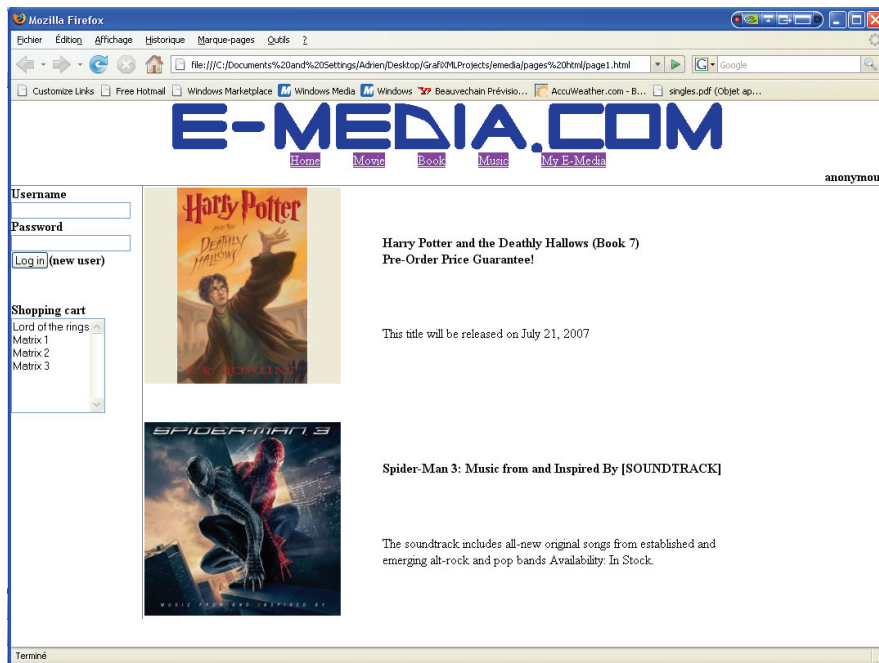
Figure 5-27 Extract of the UsiXML specifications modified with GrafiXML

Chapter 5 Surveys

```
<style>
<!--
body{ background-color:#ffffff}
.left{ text-align: left;display: block;}
.middle{ text-align: center;display: block;margin-left: auto;margin-right: auto}
*{ margin: 0;padding: 0;}
.right{ text-align: right;display: block;margin-left: auto;}
#centerb{ margin: auto 15% auto 15%;border-left: 1px solid grey}
#Label_9{ font-weight:bolder;}
#Label_7{ font-weight:bolder;}
#Label_6{ font-weight:bolder;}
#Label_5{ font-weight:bolder;}
#Label_4{ font-weight:bolder;}
#leftb{ position: absolute;top: 97px;left: -6px;width: 15%;}
#Label_3{ font-weight:bolder;}
#Label_2{ font-weight:bolder;}
#Label_1{ font-weight:bolder;}
#topb{ border-bottom: 1px solid grey }
.gridl{ display: inline-table;}
#rightb{ position: absolute;top: auto;right: 0px;width: 15%;}
.gridb{ display:table;}
...
-->
</style>
```

Figure 5-29 Extract of the cascading style sheet applied to the XHTML export

As a result of the process initiated from a low fidelity prototype perspective, we can obtain the set of web pages depicted on Figure 6-11.



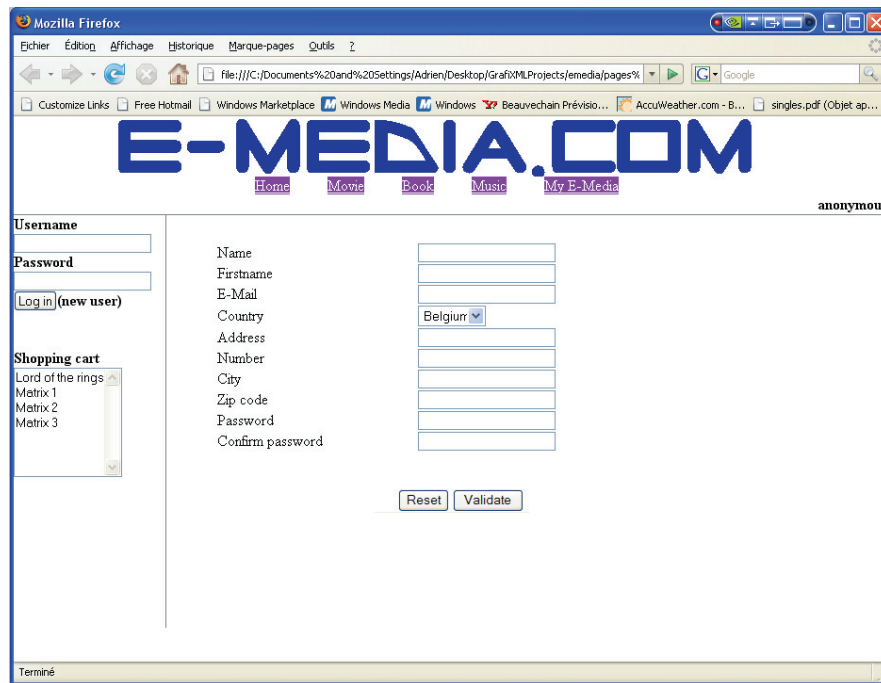


Figure 5-30 E-Media prototype rendered in XHTML with cascading style sheet

5.5.2 Find a movie

This case study proposes the development of movies finder service available for pocket pc users. This service permits to a user to search a movie using an internet connection on his pocket pc. The research can be based on a set of criteria about the type of movie, some keywords and information on the target region. Additionally, a user must have the possibility to store its preferences on the web site. Thus, the web site must integrate a web page for the account creation. As, the interaction means is not very convenient, only major information should be provided for the creation of a profile. In addition to the research feature, the user can visualize the list of the recent movies and the ranking of the best entries. Obviously, for each movie the user can visualize the information associated, and visualize the picture of the movie.

When starting the design of this PDA-based website, we define a new configuration for SketchiXML (see Figure 5-31). The first case study was using a default configuration for UI design. Here, we specify the target platform as a pocket pc, this means that the interaction surface with the application will be adapted so as to reflect the constraints associated with this platform.

Chapter 5 Surveys

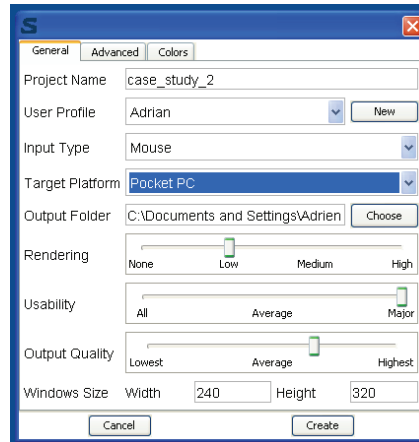


Figure 5-31 SketchiXML configured for pocket pc UIs design

Figure 5-32 illustrates the design phase for this kind of configuration. Even if the main principle remains constant with the previous example, the drawing surface is reduced to the size of a standard pocket pc screen. Accordingly, the set of constraints to be applied when testing a relation between the shapes is adapted to the current configuration. As, the size of the drawing surface is drastically reduced, the proximity between the shapes is much smaller than in a standard user interface. In order to avoid detecting wrong combination, the results of the test need to be adapted consequently.

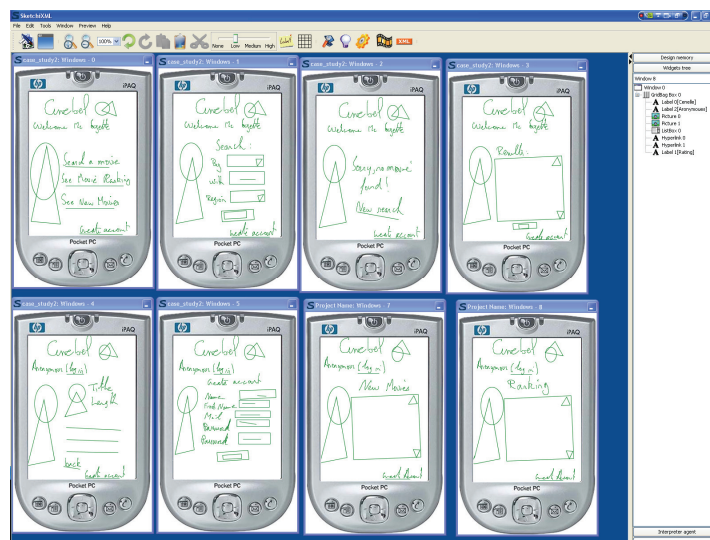


Figure 5-32 Prototyping the web site with SketchiXML configured for pocket pc design

Chapter 5 Surveys

As it can be seen on Figure 6-13 the main difficulties with such a development, is to adapt the content of the UI to the size of the window. So, contrary the previous case study where we started with a description of the navigation, we build all the web pages in a first time. Similarly to the e-media case study, it is still possible to proceed to copy paste of the common part of the web pages.

Once the content of the pages is defined, we proceed to the next step of the development by specifying the navigation between the windows. Figure 5-33 (a) illustrates the set of UI sketched in the navigation editor without any relations. Figure 5-33(b) illustrates a large part of the navigational aspect. As we can see, a same component can originate a transition to a set of UI based on a condition. Such a condition cannot be expressed in the current version of UsiXML and cannot be expressed visually on the navigation editor.

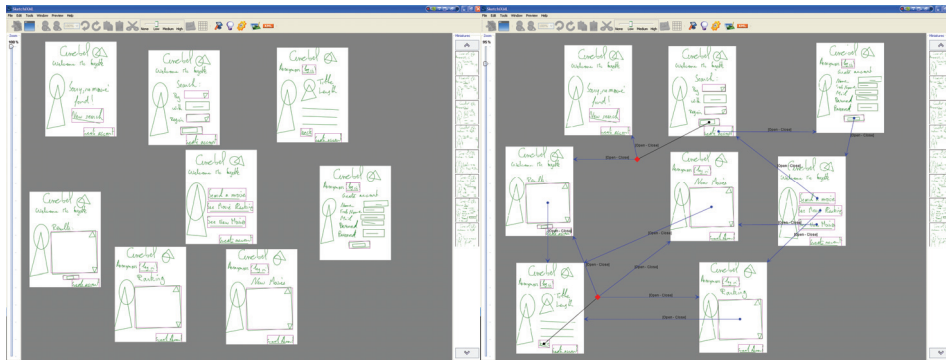


Figure 6-14 (a & b) SketchiXML navigation editor applied to the movie finder web site

Based on the two previous stages, we can now proceed to the validation of the web site thanks to the run mode. Unfortunately we do not permit to test the web site on a pocket pc, even if such an improvement could be imagined in the later version of the application. Thus, the current version of SketchiXML only proposes to the user to visualize the future web site as a sequence of frame associated to each web page.

Chapter 5 Surveys



Figure 5-34 UIs generated for the preview mode

As we explained in the previous case study, using a high level of fidelity would be confusing for a web page; this is why we prefer to use the low fidelity level. Using this perspective, the user can interact directly with the systems, and evaluate the navigation of the web site. Each time a transition involving several windows is invoked, the user has to manually specify the target windows (window 4 in Figure 5-34). Even if such manipulation does not appear very convenient, no prototyping tool offers a suitable alternative for such an issue without requiring manual coding.

Chapter 5 Surveys

```
<?xml version="1.0" encoding="UTF-8" ?>
- <uiModel id="Case_study_2" name="Case_study_2" creationDate="2007-04-03T16:16:58.109+02:00" schemaVersion="1.8"
  xmlns="http://www.usixml.org">
- <head>
  <version modifDate="2007-04-03T16:16:58.109+02:00" />
  <authorName>Adrian</authorName>
  <comment>This file was generated with SketchiXML</comment>
  <comment>Information on this tool can be found on www.usixml.org</comment>
</head>
- <guiModel id="Case_study_2-cui" name="Case_study_2-cui">
- <window id="window_0" name="window_0" isVisible="true" isEnabled="true" width="240" height="320" isAlwaysOnTop="false"
  isResizable="true">
- <gridBagBox id="Box_0" name="Box_0" gridHeight="16" gridWidth="12">
- <constraint gridx="2" gridy="0" gridwidth="5" gridheight="2" weightx="1.0" weighty="1.0" fill="none">
  <outputText id="Label_0" name="Label_0" isVisible="true" isEnabled="true" fgColor="#000000" bgColor="#ece9d8" isBold="true"
  isUnderline="false" isStrikethrough="false" isSubscript="false" isSuperscript="false" isPreformatted="false" textColor="#000000"
  textFont="Dialog" isItalic="false" visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="7" gridy="1" gridwidth="2" gridheight="2" weightx="1.0" weighty="1.0" fill="none">
  <imageComponent id="Picture_0" name="Picture_0" isVisible="true" isEnabled="true" fgColor="#000000" bgColor="#ece9d8"
  textColor="#000000" />
</constraint>
- <constraint gridx="0" gridy="3" gridwidth="8" gridheight="2" weightx="1.0" weighty="1.0" fill="none">
  <outputText id="Label_1" name="Label_1" isVisible="true" isEnabled="true" fgColor="#000000" bgColor="#ece9d8" isBold="true"
  isUnderline="false" isStrikethrough="false" isSubscript="false" isSuperscript="false" isPreformatted="false" textColor="#000000"
  textFont="Dialog" isItalic="false" visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
+ <window id="window_1" name="window_1" isVisible="true" isEnabled="true" width="240" height="320" isAlwaysOnTop="false"
  isResizable="true">
+ </guiModel>
- <contextModel id="Case_study_2-contextModel_0" name="Case_study_2-contextModel">
- <context id="Case_study_2-contextModel_0" name="Case_study_2-context-en_US">
  <userStereotype id="Case_study_2-sten_US_9" language="en_US" stereotypeName="Case_study_2-sten_US" />
  <platform id="Case_study_2-platform_0" name="Case_study_2-platform">
    <hardwarePlatform category="pocket pc" keyboard="virtual" pointingDevice="Stylus" screenWidth="240" screenHeight="320"
    hasTouchScreen="true" />
  </platform>
  <environment id="Case_study_2-env_0" name="Case_study_2-env" />
</context>
</contextModel>
+ <resourceModel>
</uiModel>
```

Figure 5-35 UsiXML specifications generated by SketchiXML

Once the web site is validated, the designer can generate the corresponding specification in UsiXML (Figure 5-35). Naturally, the structure of the file generated is very similar to the previous case study. All the components are specified with default value and based on a grid bag layout. The main difference lies in the size of the window and to the fact that a context is associated to the UsiXML file. This context (surrounded by a rectangle on Figure 5-35) specifies that the file was originally developed for a pocket pc with a screen resolution of 240 * 320 pixels. The context can store a lot more information than this small subset of attributes, but as it is already the case for the widgets; SketchiXML only generates default value for some key attributes. The complete list of attributes that can be present in a context model can be found in Appendix A. Then, based on this information, other UsiXML-based tools can adapt this set of UI for other context.

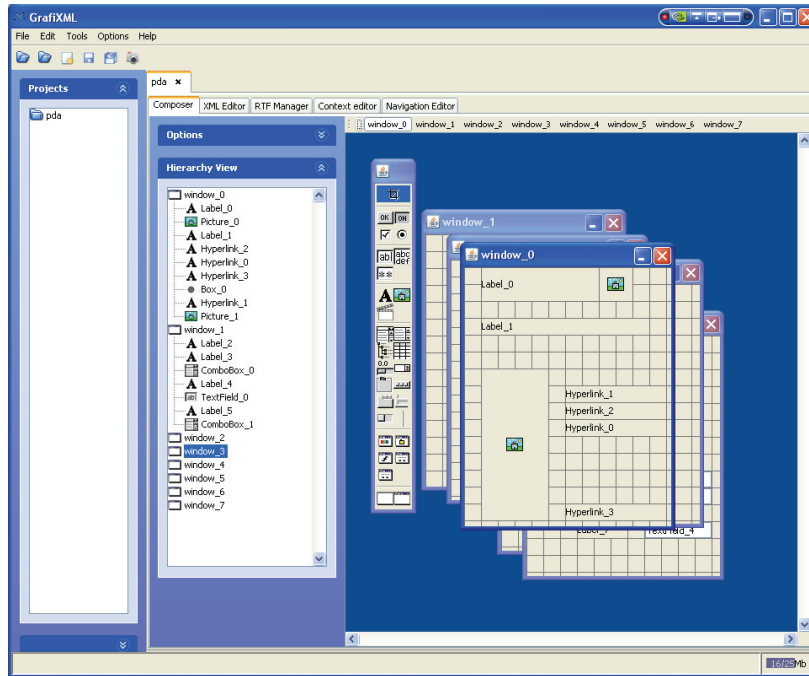


Figure 5-36 Specifications generated by SketchiXML imported in GrafiXML

The UsiXML is then imported in GrafiXML for enhancements. All the details that cannot be and must not be specified in the first stage of the prototyping process are now developed. Using GrafiXML the designer has no visual information on the target platform such as proposed in SketchiXML, but on the other hand, the designer does not necessary need this kind of information as he is supposed to be used to UI design. Figure 5-36 shows the specification produced by SketchiXML imported in GrafiXML.

The output produced by GrafiXML consists in a new UsiXML file where all the values that were initially by default are filled according to the designer preferences. Such file is very similar to the output presented for the previous case study; there is thus no interest in showing the result. Figure 5-37 provides an example of the XHTML code generated by GrafiXML. Similarly to the previous example, all the components' attributes are specified in a cascading style sheet.

Chapter 5 Surveys

```
<body>
<div id="topb">
<div style="text-align:center"><span class="Style1" id="Label_0">Cinebel</span>
</div>
<div style="text-align:center"><span id="Label_1">Welcome Mr Coyette</span></div>
</div>
<div id="left">
<table >
  <tr>
    <td width="79" rowspan="6"></td>
    <td width="2">&nbsp;</td>
    <td width="142">&nbsp;</td>
  </tr>
  <tr>
    <td>&nbsp;</td>
    <td><span class="Hyperlink_1">Search a movie</span></td>
  </tr>
  ...

```

Figure 5-37 XHTML generated with GrafiXML

Figure 5-38 present the web page obtained from the early prototype presented in Figure 5-32. Based on the specifications generated as output by SketchiXML and GrafiXML, other UsiXML tools permit to transform the set of web page for other computing platform. The complete list of tool can be found in Appendix B.



Figure 5-38 Visual representation of XHTML Mobile Profile files generated by GrafiXML

Chapter 5 Surveys

5.5.3 Designing a wizard

This small case study propose another context for the development of the prototype: we develop a wizard in Java. The wizard is composed of a sequence of 6 screens aiming at capturing the information to be used by the application.

The first screen only provides some information on the wizard and the way parameters can be changed afterwards. The second screen capture information on the designer and the project. The third frame captures information on the plug-ins to be integrated to the current project. Fourth, we ask the designer to choose a set of languages to be supported by the UIs to be designed. Fifth, the wizard ask the designer to choose a template amongst a lit of predefined templates. Eventually, the last UI confirm that the project is ready to be designed.

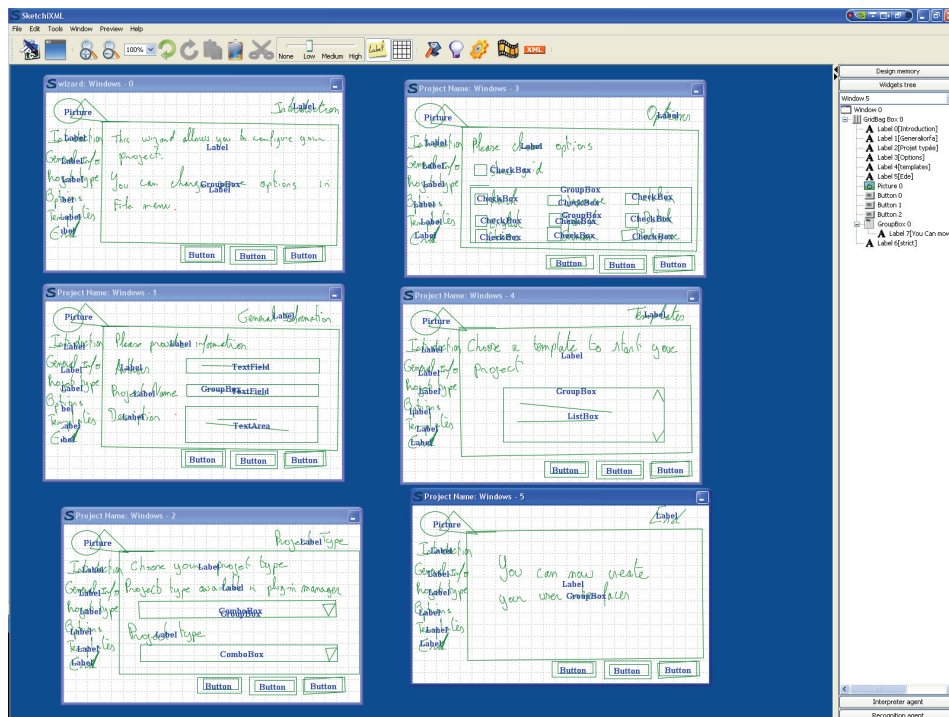


Figure 5-39 Sketching the wizard with SketchiXML

Oppositely to the previous examples, the output to be produced is not a web page; we can thus consider using high fidelity representation of the prototype without perturbing the end user with a representation that looks final but far from the expected result. As we can observe on Figure 5-39, we specify the global

Chapter 5 Surveys

navigation based on the prototype with a high level of fidelity. Obviously, the global navigation appears very simplistic, as the only navigational aspects that can be represented in this case study, consist in “next” and “previous” transition between the dialog boxes part of the wizard.

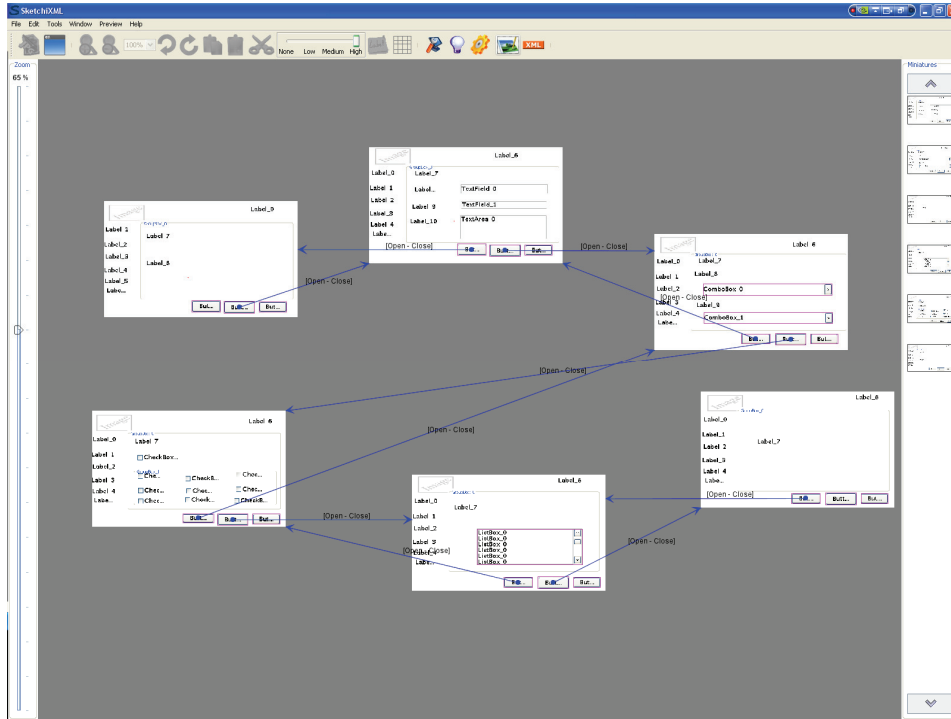
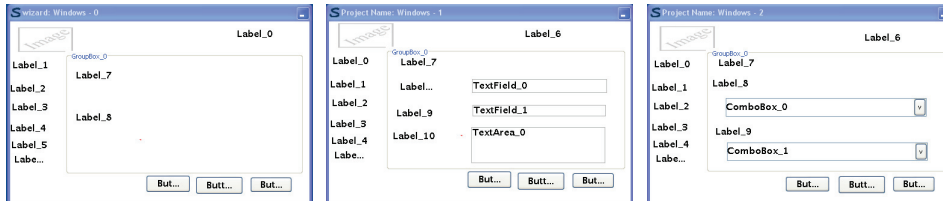


Figure 5-40 Specifying the navigation with SketchiXML

Based on the navigation definition, the designer has the opportunity to evaluate the prototype thanks to the run mode. Oppositely to the previous case studies, we use the high fidelity representation for the run mode; we obtain thus a set of UI based on the look and feel installed on the machine. Obviously, the designer is always free to use one of the other fidelity level according to its needs or preferences.



Chapter 5 Surveys

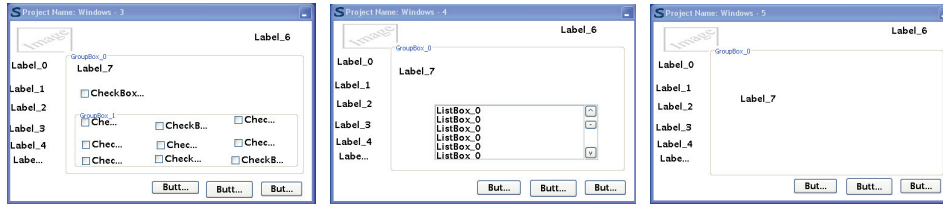


Figure 5-41 Testing the prototype thanks to the run mode

As a result of the prototyping phase, UsiXML specifications are generated (Figure 5-41). As we observed in the two previous example, the sketch only capture the core properties of the UIs. All the components are specified with default values for their attributes.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <uiModel xmlns="http://www.usixml.org" id="Wizard" name="Wizard"
  creationDate="2007-05-14T11:06:57.187+02:00" schemaVersion="1.6.4">
+ <head>
- <cuiModel id="Wizard-cui" name="Wizard-cui">
- <window id="window_0" name="window_0" isVisible="true" isEnabled="true"
  width="500" height="299" isAlwaysOnTop="false" isResizable="true">
- <gridBagBox id="Box_0" name="Box_0" gridHeight="0" gridWidth="0">
- <constraint gridx="1" gridy="0" gridwidth="5" gridheight="2" weightx="1.0"
  weighty="1.0" fill="none">
  <imageComponent id="Picture_0" name="Picture_0" isVisible="true"
    isEnabled="true" fgColor="#000000" bgColor="#ece9d8" textColor="#000000"
  />
  </constraint>
- <constraint gridx="5" gridy="2" gridwidth="20" gridheight="10" weightx="1.0"
  weighty="1.0" fill="none">
- <groupBox id="GroupBox_0" name="GroupBox_0" isVisible="true" isEnabled="true"
  fgColor="#000000" bgColor="#ece9d8">
- <gridBagBox id="GroupBox_0_Box" name="GroupBox_0_Box" gridHeight="10"
  gridWidth="19">
- <constraint gridx="0" gridy="1" gridwidth="18" gridheight="3" weightx="1.0"
  weighty="1.0" fill="none">
  <outputText id="Label_7" name="Label_7" isVisible="true" isEnabled="true"
    fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false"
    isStrikethrough="false" isSubscript="false" isSuperscript="false"
    isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
    visitedLinkColor="#000000" textMargin="0" numberOfColumns="15"
    numberOfLines="1" />
  </constraint>
- <constraint gridx="0" gridy="3" gridwidth="18" gridheight="4" weightx="1.0"
  weighty="1.0" fill="none">
  <outputText id="Label_8" name="Label_8" isVisible="true" isEnabled="true"
    fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false"
    isStrikethrough="false" isSubscript="false" isSuperscript="false"
    isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
    visitedLinkColor="#000000" textMargin="0" numberOfColumns="15"
    numberOfLines="1" />
  </constraint>
  </gridBagBox>
- </groupBox>
- </gridBagBox>
```

Figure 5-42 Sketching the wizard with GrafXML

Chapter 5 Surveys

Figure 5-43 illustrate the importation of the SketchiXML output in GrafiXML. The specifications obtained from the low fidelity prototyping phase can then be exploited in GrafiXML. We can refine those specifications in order to obtain very precise descriptions of the UIs. All the components attributes are defined and the UsiXML specifications are updated accordingly.

Based on these specifications, we can generate the corresponding code in Java. Contrary to the two previous examples where the code generation required a manual intervention for the layout conversion, the interpretation in java is straightforward. Indeed, as it was presented earlier in this thesis, the layout mechanisms used in UsiXML are almost similar to the layout in Java.

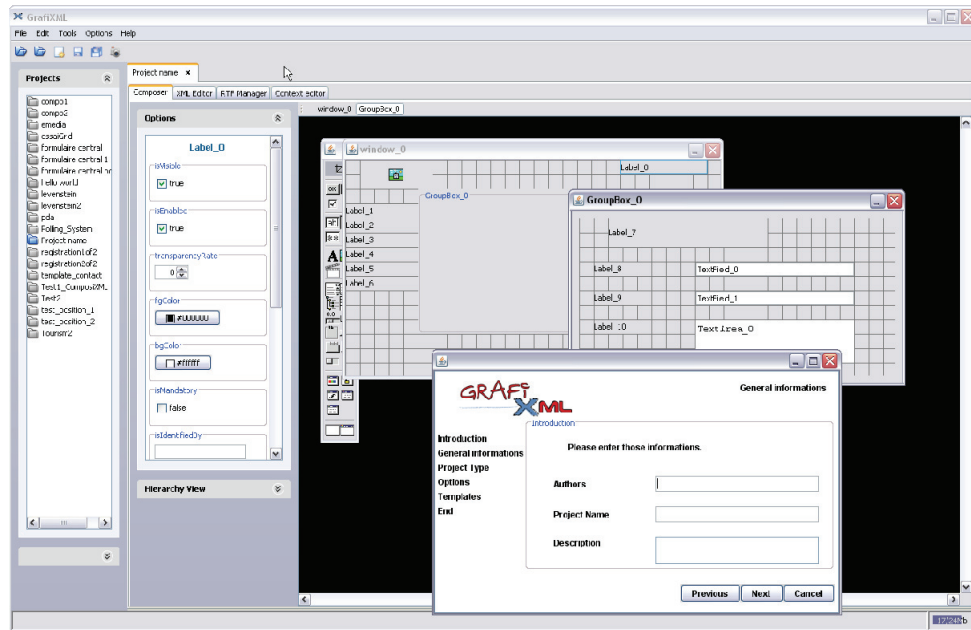


Figure 5-43 Wizard imported in GrafiXML and the “General Information” frame rendered in java

Figure 5-44 provides a short extract of the java code generated as a result of the prototyping process. We can observe that the layout that was adopted by SketchiXML is clearly similar to the layout of the java class. Indeed, all the components present on the java file generated are positioned on a grid bag layout. As a consequence, the extract proposed in 5-44 is almost unreadable. Even if such layout permits to build very complex UI, the corresponding code is often very long as many constraints need to be specified.


```
public class Wizard {
    static class Window_0 extends JFrame {
        public Window_0() {
            super(getText("window_0", CONTENT));
            setDefaultCloseOperation(EXIT_ON_CLOSE);
            GridBagLayout grid = new GridBagLayout();
            GridBagConstraints c = new GridBagConstraints();
            getContentPane().setLayout(grid);
            c.fill = 0;
            c.gridheight = 2;
            c.gridwidth = 5;
            c.gridx = 1;
            c.gridy = 0;
            Picture_0 = new JLabel(getText("Picture_0", CONTENT));
            Picture_0.setToolTipText(getText("Picture_0", TOOLTIP));
            getContentPane().add(Picture_0);
            grid.setConstraints(Picture_0, c);
            Picture_0.setForeground(new Color(Integer.parseInt("000000",
16)));Picture_0.setBackground(new Color(Integer.parseInt("ffffff", 16)));
            c.fill = 0;
            c.gridheight = 10;
            c.gridwidth = 20;
            c.gridx = 5;
            c.gridy = 2;
            grid.setConstraints(Picture_0, c);
            c.fill = 0;
            c.gridheight = 1;
            c.gridwidth = 5;
            c.gridx = 0;
            c.gridy = 3;
            Label_1 = new JLabel(getText("Label_1", CONTENT));
            Label_1.setToolTipText(getText("Label_1", TOOLTIP));
            grid.setConstraints(Label_1, c);
            getContentPane().add(Label_1);
            if (! getText("Label_1", ICON).equals(""))
Label_1.setIcon(new ImageIcon(this.getClass().getResource(getText("Label_1", ICON))));
            Label_1.setFont(new Font("Dialog", 1, 12));Label_1.setForeground(new
Color(Integer.parseInt("000000", 16));Label_1.setBackground(new Color(Integer.parseInt("ffffff", 16)));
            c.fill = 0;
            c.gridheight = 1;
            c.gridwidth = 5;
            c.gridx = 0;
            c.gridy = 4;
            Label_2 = new JLabel(getText("Label_2", CONTENT));
            Label_2.setToolTipText(getText("Label_2", TOOLTIP));
            grid.setConstraints(Label_2, c);
            getContentPane().add(Label_2);
            if (! getText("Label_2", ICON).equals(""))
Label_2.setIcon(new ImageIcon(this.getClass().getResource(getText("Label_2", ICON))));
            Label_2.setFont(new Font("Dialog", 1, 12));Label_2.setForeground(new
Color(Integer.parseInt("000000", 16));Label_2.setBackground(new Color(Integer.parseInt("ffffff", 16)));
            c.fill = 0;
            c.gridheight = 1;
            c.gridwidth = 5;
            c.gridx = 0;
            c.gridy = 5;
            Label_3 = new JLabel(getText("Label_3", CONTENT));
            Label_3.setToolTipText(getText("Label_3", TOOLTIP));
```

Figure 5-44 Extract of the java code generated as output

5.6 User Testing Limitations

Through the several case studies, we have demonstrated the purpose of SketchiXML and its integration with other UsiXML tools. But these case studies may appear too simple and too small to validate the approach. Indeed, a large scale application would be more appropriated. Ideally, two case studies should be developed simultaneously, one with the standard tools and approaches while the second would be based on the framework depicted in this thesis.

Unfortunately, such validation is impossible in the current context. First of all, I doubt any industrial would be ready to consider this approach seriously, time is money and a several employees spending time for an experiment (without financial return) is likely to discourage any employer. Secondly, SketchiXML and the other tools of the framework contain a series of small bugs. Indeed, as most of the tools developed in the scientific community, the tool is closer to a good prototype than an industrial version. Consequently, the major comments that would be collected from such an experiment would be related to the unfinished aspect of the applications rather than their qualities.

This problem could hardly be solved in the current context; to address this issue years of development would be required before SketchiXML and others tools can pretend to be ready for industrial testing. Unfortunately we do not have the human and financial resources to conduct such a development.

However, we can image the test procedure that could be conducted if such resources were made available. On one hand, we could evaluate each individual tool part of the framework and asses their respective performances. As we have presented earlier in this chapter with the surveys for SketchiXML, such evaluation can be done by comparing the performances with the expectations, and by collecting user feedbacks on the applications.

The evaluation of the complete methodology is more complicated as the framework is a complement to methodologies and not a methodology itself. Obviously, if a comparison need to be done between the approach that integrate the framework, the comparison should only focused on the factors specific to the framework and not the methodology used. The following list, give an overview of criteria to be considered grouped in two categories inspired from [Arth86]:

Objectives:

- Early error detection
- Traceability
- Readability
- Ease of change
- Reduced complexity
- Speed
- Enhanced cohesion
- Well-defined interfaces

Attributes:

- Adaptability
- Portability
- Reliability
- Testability
- Reusability
- Correctness
- Maintability

Based on the observations made for each of these criteria, we would have all the relevant information needed for a complete comparison of both approaches.

5.7 Conclusion

Through this chapter we have presented four studies that were conducted during the last years. As presented in the first section, some results were very concluding from the very beginning while other needed to be improved. Indeed, the first survey showed good usability results while the performances of the application were far below the expectations.

With the next surveys that were conducted two years later, we observed that the performances of the application were considerably closer of the expectations, even if the results need to be interpreted with care. Contrary to the first survey, the examiner was not in the same room than the participant, also the purpose of the experiments was different and many error can be attributed to the test configuration. Even if we do not have any statistics to illustrate this problem, we observed that many participants did not read carefully the instructions and sketched the wrong widget. Based on this observation we cannot compare precisely the value provided by both surveys as the test condition were not the same. However, despite the different test conditions, we observe that the recognition rate obtained during the last survey is far better than the result obtained previously. This point is very positive as we consider that the second survey was more complicated than first set of test.

This chapter has also presented the construction of the grammar to be used by the application. The purpose of the experiment was to define a grammar that

Chapter 5 Surveys

would be really intuitive for the designers. The version of SketchiXML used at that time did not permit to define custom representations for the widgets, it was thus very important that the predefined representation were well defined. The next survey addressing the representation issues occurred in a different context, SketchiXML had another interpretation mechanism based on an xml grammar. Not only the grammar was easily editable, but many representations for a single widget could be added with a lot of flexibility. Also, SketchiXML had a new feature allowing to switch from a fidelity level to another. The purpose of the second survey was thus to evaluate the quality of the recognition with the new interpretation mechanism, the impact of the fidelity level, and the impact of the use constraint in widget representation.

We observed that the fidelity level had no impact on the construction of individual widget. This observation must be used carefully as it does not mean that the fidelity level has no impact on the design of complete UI. The third survey was aimed at evaluating this aspect, but participant did not follow the procedure correctly. Indeed, participants were asked to sketch UI in a particular level of fidelity, but most of them did not respect it. The results were thus useless, but, based on this issue we developed the following surveys in so as to avoid this kind of problems.

The survey on the widget representations revealed some interesting points; we have observed that the number of constraints in a representation significantly increase the widget complexity. Unsurprisingly, some constraint have appeared to be more complex than other, but some constraint that were considered to be very soft appeared very difficult (see toggle button). This reason is likely to found in an error in the java code than the intrinsic complexity associated to the constraint.

Through this chapter we have provided three case studies using SketchiXML and others tools of the UsiXML suite.

With the first case study, we have illustrated the design of an e-commerce web site. Though this example, we have illustrated the complete process starting from a low fidelity prototype to a high fidelity prototype. To this end, illustration of the UsiXML specifications refinement from a low fidelity description to a high fidelity description was provided. Based on this rich UI description, we generated the corresponding UIs in XHTML.

Chapter 5 Surveys

The second example was more concise as it focused on the design of UIs for a pocket pc. To this end, the case study illustrated how the tools can be adapted so as to illustrate the new constraint associated to this kind of development. Based on the UsiXML specification generated as output, we produced a set of web pages to be displayed on such device.

The third example provided presented the development of an application for a third context of use. Through this example we developed a java wizard based on UsiXML specifications. As the previous case studies, we started from a low fidelity mock up of the expected result and we refined it with GrafiXML in order to obtain complete UIs.

Chapter 6 Conclusion

This chapter concludes the work of this thesis by reminding its context (Section 1), summarizing the original contribution of this thesis (Section 2), and by outlining how this work has been validated (Section 3). A list of individual contributions of this thesis is first provided (Section 4) and major expected avenues for future work are suggested (Section 5).

6.1 Context of This Work

Given the importance of user interface design during the application development life cycle, it has been realized that much effort remains to be done on the development of a new technique to reduce the design time needed to obtain the right user interface.

For this purpose, many UI interface builders came on the forefront for most of the existing programming and markup languages, allowing building a final UI faster. Unfortunately, even if these tools turned out to be very efficient for building a final UI, designers were still looking for a precise methodology to proceed to the construction of the future UIs that is explicitly supported by a prototyping tool exhibiting several expanded capabilities such as ease of use, fast turn-around, designer control, prototyping captures.

As a response to the failure to rationalize the user interface process, initiatives aimed to introduce new development methods and/or tools to effectively and efficiently support UI design, and particularly early design which was largely underexplored. Paper and pencil appeared as one of the very first and most

Chapter 6 Conclusion

effective ways to represent the first drafts of a future UI. Indeed, this kind of familiar and unconstrained approach benefits from many advantages since sketches can be drawn during any design stage, it is fast to learn and quick to produce, it lets the sketcher focus on basic structural issues instead of unimportant details, it is very appropriate to convey ongoing, unfinished designs, and it encourages creativity. Last but not least, sketches can be performed collaboratively between designers and end-users, or even other stakeholders if needed. The idea of developing a computer-based tool for sketching UIs naturally emerged from these observations trying to combine the best of both the hand sketching process and a computer assisted interface design. Since it was observed that the full potential of such techniques was not yet reached in existing tools, a new sketching tool, named SketchiXML, has been developed with the purpose to solve the shortcomings identified in those existing software.

This sketching tool has extended a set of UsiXML-compliant tools providing designers and developers with more support to the prototyping framework. Indeed, prior to this activity, no set of tools offered such a similar flexibility for the design process: many independent tools exist, but each of them addresses only a specific issue of the prototyping process and they cannot necessarily be combined in a genuine interoperability.

Based on this framework, this thesis illustrates how combining some existing development methodologies with the set of tools part of the framework presented may provide a more extensive support to UI early design. Indeed, rapid prototyping is recognized for its valuable contribution in software development, but is not always used as such for many reasons (e.g., it is a waste of time since its output cannot be reused for another project). To this aim, this thesis progressively addressed a series of open questions for which some answers have been given. The outline of these contents is summarized in the next section.

6.2 Content of This Dissertation

Through the state of the art of Chapter 3 revealed a series of shortcomings in existing approaches for achieving computer assisted UI design. These shortcomings delineated our problem space and lead us to conclude that a future sketching tool could be improved along several dimensions. Fifteen various development methodologies were surveyed and compared with respect to their conceptual coverage along with their respective design process.

Chapter 6 Conclusion

Chapter 3 presented the SketchiXML agent oriented architecture developed to support the series of aforementioned requirements: the strategy used for the shape recognition and its interpretation mechanism, a grammar mechanism for UI widgets and their representations based on experimental results, its working principle and the functionalities of the sketching tool.

Through Chapter 4, a prototyping framework has been examined based on a series of interoperable software based on a common User Interface Description Language: UsiXML. Then, a selected set of existing methodologies has been explored and exemplified on how they could be augmented with prototyping capabilities provided by a sketching tool.

Chapter 5 demonstrated how human aspects related to sketching activities as supported by SketchiXML have been considered. Various surveys have been carried out for exploratory and evaluation purposes. The first phase of surveys evaluated the performance of the application and collected user feeling about their experience with the tool. The second phase consisted of evaluating the performance with respect to the previous tests, but also of evaluating the potential impact of the fidelity level used for the sketching phase. Two aspects were taken into account for this experiment: the construction of an individual widget and the construction of a complete UI. A CSUQ test concluded the usability study of the sketching tool. We also illustrated with concrete case studies how SketchiXML could be effectively used to develop a XHTML web application and a Java stand-alone interactive application.

6.3 Validation

6.3.1 External Validation

The external validation was realized through the series of user tests of SketchiXML conducted on a reasonably large sampling of participants. The main goals of these tests were to evaluate the feasibility of the approach; firstly, by evaluating the technical feasibility of this sketching tool, and above all by evaluating its capability to involve people with various background in the design process. To this end, it was judged of crucial importance that participants had a good feeling of the application and were able to produce significant results within a short amount of time.

Chapter 6 Conclusion

Chapter 5 demonstrated the feasibility of involving participants with various backgrounds in a single setup: all types of group showed good ability with the application. Moreover, the participants without any past experiment in UI design surprisingly showed better result than the group of computer scientist. This unexpected difference was the proof that the tool was suitable for different kinds of user.

The second positive effect was the learning curve. Indeed, two tests measuring the learning effect of the participants were conducted, both leading to the same observation. The first of the two tests consisted of designing three windows, while the first and the third were similar. As presented in Chapter 5, the results of the survey were very positive; even if the recognition rate was the same for each of the UIs sketched, the time needed to draw the first and the third one were significantly different, the time required for the third one being smaller. The second test aimed at testing this aspect was done with “12x4x2” test. In this test we also observed that the performance of the participant are significantly better and the end of the test. So, after a ten minutes period, the performance of the participant were already clearly higher. So, we consider that if a designer can learn how to use the application in very short delay, then the application would gain credibility in its purpose to combine the advantage of paper prototype and computer assisted design.

Additionally to an easy learning, it was important that participants had a positive feeling vis-à-vis of the application. To this aim, a part of the surveys were dedicated to this purpose as we collected information about their experience. On one hand we asked to the participants to fill a short questionnaire about the application and we proceeded to an interview. Even if most of the participant were really pleased and enjoyed the use the application, we captured many issues that could be enhanced in the next release of the tool. The overall feeling of the participant was quite positive as many enjoyed the test; most of them were ready to re-use SketchiXML in the future.

6.3.2 Internal Validation

The internal validation of a methodology consists in assessing its characteristics against a set of selected criteria. The relevant criteria, called requirements, for our methodology have been elicited and motivated after the state of the art of Chapter

Chapter 6 Conclusion

3. This section revisits the list of these requirements in the light of the work conducted in this thesis:

- R 1.** *Avoidance of Effort loss.* Some sketching tools only support the sketching activities without producing any re-usable output. So, once the designer and the end user agreed upon a sketch, a contract can be signed between them and the development phase can proceed from the early design phase to the next phases. When the sketch is not transformed, the effort building it is lost. A better alternative would be to consider a mean to re-use the output in an efficient manner and avoid any effort or time loss.

Discussion: As presented through Chapters 3,4,5, the SketchiXML output can be reused in most of the UsiXML based tools. As shown in Chapter 7, the work that was done in the early prototyping phase with SketchiXML can be re-imported in a high-fidelity editor such as GrafiXML so as to define all the aspects that remain to be specified after early design.

- R 2.** *Well defined editing functionalities.* The purpose of our tool is to combine the advantages of the paper prototyping and the computer assisted design. As the main advantage of computer assisted design seems to be the possibility to easily move, copy, paste, zoom, undo... these functionalities should be present in such a tool in a well defined way. Such assertion seems very natural, but all the tools do not always propose such functionalities, or just a small subset. Since a design project is likely to involve more than one single UI at a time, it is also necessary for a tool to support multi-windowing design. Other significant and obvious functionalities are the import and export functions. It is very important to have the opportunity to save the current work and reopen it later, even on a different computer

Discussion: As stated in many occasions throughout this document, we aim at truly combining the advantages of both, the computer assisted design and the paper based approach. As a consequence, effort was done in order to propose support for general edit functions. Chapter 4 provided a description of the functionalities that are supported by the tools. All the functions that were elicited in the Requirement 2 are

met. Not only SketchiXML supports copy, paste, cut, undo, delete, redo and zoom, but it also permits to save the current work or to export it to a binary file or to a UsiXML file.

- R 3.** *Language neutrality.* Most of the times, when a sketching tool support code generation, it is bound to a particular programming language, a particular UI type, a particular computing platform or operating system. So, once an output is produced, it is usually bound to one particular environment, therefore preventing developers to reuse sketches from one case to another, such as for various platforms. As the number context of use is increasing extremely fast, being bound to a specific platform is a clear handicap these days. So, in order to meet the designer's need, the prototyping tool should provide an output, that is general and context independent. For this purpose we recommend the use of a specification language for UI description. Several of them were developed these last years addressing this challenging issue.

Discussion: As response to this requirement, we have decided to consider the exportation of the work to an abstract description language for user interface. So, SketchiXML does not generate user interface in a specific programming language but in UsiXML specification instead. The output generated is thus independent from any programming language and computing platforms. However, the specifications generated by the tool are associated to a specific context of use that will be used by the other UsiXML based tool for adaptation for other contexts of use.

- R 4.** *Robust recognition.* If a tool supports shape or text recognition, the recognition quality and rate should to very high so as to prevent the designer to waste time with misrecognition, thus leading to discouragement and disappointment. If a designer should rewrite the text several times before it is recognized, this feature should be either disabled or improved. Another consideration for text recognition would be to hide the result from the designer during the process, so even if it was not recognized properly, the designer is not tempted to delete and rewrite it.

Discussion: Thanks to the use of the combination of Cali library SketchiXML was able to recognize strokes of different sizes, rotated at arbitrary angles, drawn with dashed, continuous strokes or overlapping lines with a high recognition rate. But the results of the first study showed an average recognition rate smaller than 90 percent. The next survey used a new version based on combination of the Cali library and the gesture recognizer described in section 4. In addition, the test was conducted with an improved UI and a large pen-enabled device. Such configuration boosted the performance in term of recognition as we reached 92.5%. Even if such recognition rate could still be improved, we consider it a satisfactory with regards to the test conditions that were not optimal for testing such aspect.

- R 5.** *Large conceptual coverage.* When considering denim or other similar tools, the conceptual coverage is not a problem since there are not any kinds of constraint on the drawing. For the other kind of tool the situation is different, each tool must specify a set of representations for each widget. These representations can be based on a shapes combination, a single gesture or a mix. Most of the sketching tools providing shape recognition only support a small set of widgets, preventing to build any complex user interfaces. Moreover, adding new representation is very difficult since most of the tools do not provide any functionality to enrich the grammar. Also, if the designer has the opportunity to add a representation, i.e., a new gesture associated to a widget, adding a new widget is always impossible.

Discussion: SketchiXML is able to recognize a very large set of widgets. Thanks to its easily editable grammar, the designer has the possibility to add new widgets or to specify new representations for the existing widgets. To this aim a set of constraint is defined by default in the tool, but can be edited easily even if manual coding is required. Moreover, UsiXML permit to save custom widgets as generic widgets. Such characteristic is very interesting as the designer is free to add new widget to maintain a mapping between the user interface sketched and its corresponding view in UsiXML.

Chapter 6 Conclusion

R 6. *Recognition and process flexibility.* Most of the sketching tools providing shape recognition try to recognize every stroke drawn (either in batch or real time mode). This prevents the user to represent complex illustration on the future interfaces, such as diagrams, that cannot be represented as widget. Moreover, this constraint also exists for the high-fidelity design tools. For instance, Figure 8-1 illustrates a UI that could not be represented with existing low- and high-fidelity design tools providing shape recognition.

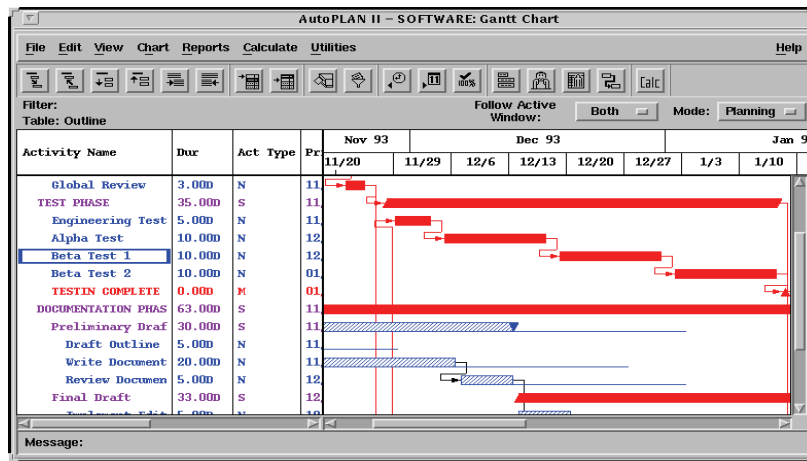


Figure 6-1 An example of user interface that cannot be designed with a standard UI builder
(Source: [Szek96])

In addition to a flexible recognition, the interpretation process should also hold a major role. UI Sketching tools do not allow a lot of flexibility in the sketch process: the user cannot choose when recognition will occur, degrading openness [Sumn97] and when this occurs; it is difficult to return to a previous state. Moreover, depending of the tool, each time a component is recognized, it is sometimes replaced by a stroke of a different color, left as it was drawn, replaced with a smooth representation of the component... According to the authors of these tools, their representation is the most relevant, even if no research tried to confirm such assertions. For our point of view, we should leave this decision to designer and his preferences.

Discussion: SketchiXML addresses this requirement with a series of dispositions. The fact that the designer could decide to keep a part of

the sketch a sketch that should not be recognized is supported by the system. Indeed, the designer has the possibility to switch to a free drawing mode, where sketches will be stored as pictures. Regarding the way the sketch is interpreted, we offer a very innovative solution as we permit to the designer to choose the rendering to be applied. As presented in chapter 4, the designer can choose to keep the drawing as it was sketched (none fidelity) or to display the widget recognized in one of the three fidelities supported (low, medium, high).

- R 7.** *Design history.* As it can be seen in some professional tools, the use of the design history can be very useful. When prototyping, a designer will tries to explore many design alternatives in a short time, so the designs will evolve very fast. Looking back to the previous steps can be very useful, but such functionalities is rarely supported by the existing tool fir rapid prototyping of user interface. In addition to the set of editing functionalities such function would be very useful in this kind of tool.

Discussion: Through Chapter 4 we presented the design memory feature present in SketchiXML. This feature permits the designer to store all the changes that were done on the user interface. So, the designer can visualize the window at the different stages of development and restore the current drawing to a previous state.

- R 8.** *Expressive scenario editor.* As stated in the introduction, one of the major drawbacks of the paper prototyping is the difficulty to represent the interaction between the windows. So, we consider that a good prototyping tool should support this feature, since this kind of information can easily be provided by the end user and is important for a global comprehension of the user needs. But all the sketching tools supporting code generation lack of a robust scenario editor.

Discussion: as presented in Chapter 4, SketchiXML contains a navigation editor. This editor allows representing the navigation between UIs contained in the project. Unfortunately, the navigational aspects are not sufficiently developed in the current version of UsiXML, a later version should address this issue in a better way. Even if the specification cannot be stored in the UsiXML export, this

feature is still very useful as it is used for the run mode of the application for simulation purposes

- R 9.** *Preview (Run-mode).* One of the drawbacks of the paper-based prototyping is the difficulty to switch from the design phase to a preview or a run mode. The standard approach requires a designer to play the computer and move the window accordingly to the user actions. So, if the tool is equipped with a navigation editor, we can use all the information provided by the end user and build a run mode based on the sketches or the windows interpreted in a specific programming language. Such feature is very interesting since it permit to see how the end users interact with the early prototype.

Discussion: SketchiXML offers a run-mode to the designer. This perspective permits to the designer to evaluate the navigation that was specified in the navigation editor. The run-mode works with several level of fidelity. Indeed the high-fidelity level consists in rendering the current sketches as java user interfaces. As the designer is free to develop user interface for other platform such as website, he also has the possibility to execute the run mode with a lower level of fidelity in order to avoid confusion.

- R 10.** *Ease of use (naturalness).* The key argument for the development of such tool is the ease of use. Everybody agree on the fact that paper prototype is fast, easy and do not require an extensive background in computer science. To this end, if a tool is supposed to capture the advantage of both computer assisted design and paper prototyping, the main advantage of paper prototype must have a central role in the development of the application. So, the tool must be easy to use, use only natural notation, do not impose any constraints on the sketching... Otherwise a learning curve may prevent the end users from learning how to use the tool and efficiently using it.

Discussion: The main objective of the application was to be user friendly in order to involve the end-users in the development process. To this end we tried to reduce the number of priority constraints, and to define the most natural representations for each of the widget. With regards to the results of the surveys, it seems that the objectives are

met. Indeed, almost all the participant found the application easy to use, easy to learn and enjoyed the experiment.

6.4 Contributions

With SketchiXML we have introduced a new and innovative tool. Firstly, SketchiXML is the first informal design tool that generates a user, platform, and environment independent output and thus provides a solution to the language neutrality weakness of existing approaches. Secondly, amongst all the tools supporting shapes recognition and interpretation, it is probably on the most flexible to our knowledge. Not only, SketchiXML enables designers to edit the grammar of the supported widget (this editing of this graphical grammar is totally separate and independent of the recognition engine), but it also allows adding any custom widget and keep the mapping with the output produced. In addition to this shape interpretation mechanism, SketchiXML integrates a mix of library and algorithm for shape recognition. Such an approach extracts the best of the different techniques. In order to define how the application components should be coordinated, an architecture based on a standard pattern with the model-view-controller was developed. Then, a set of agents dedicated to the shapes recognition and interpretation have been developed. The approach provided in this document permit to conduct a large set of simultaneous tasks while keeping a quick response time.

We have shown that SketchiXML meets requirements most of the requirements that were identified as important shortcomings of existing tools.

Through chapter 2, we have presenting the prototyping techniques and proposed a prototyping framework for application and user interface development. Based on this reference framework, the showed how SketchiXML extended a set of tools based on UsiXML, allowing to initiate the prototyping process from the early design phase to the final concrete user interface, with tools support for every stages. Thus supporting the approach proposed in the prototyping reference framework presented in chapter 2.

Using this prototyping framework based on the UsiXML tools, we have proposed a series of recommendations to integrate this framework into the existing methodologies. Together with GrafiXML and VisiXML, SketchiXML complete a prototyping framework based on UsiXML initiating the prototyping process from

Chapter 6 Conclusion

different perspectives. Based on this prototyping framework, chapter 6 illustrates how to integrate it to the existing software development methodologies. To this end, we proceeded to a description of general good practice for systems prototyping and we showed how to apply it to some existing methodologies for software development. The prototyping framework was thus proposed as a add-in for most of the existing methodologies, as it effectively supports many of the issues associated with prototyping techniques.

Through this thesis numerous empirical surveys were conducted. The first one was conducted on sample of 60 participants coming from different activity sectors with different backgrounds. They were solicited for the construction of the grammar. In order to identify natural representation for each of the widget, we asked these people to provide intuitive representation for the widgets to be handled by SketchiXML. As it was presented in chapter 4, this set of representation is not hard coded and can be reconfigured by the user through an external xml-based grammar. Thus, any other representation can be added easily using the grammar edition module present in SketchiXML, or by editing the grammar file directly.

The second survey aimed at evaluating the application in term of performances and usability. We conducted a survey on 40 people with different background in order to test these parameters. From these result we have identified a set of issues that should be addressed in the next releases, and we created an historical record of usability benchmarks for future releases.

The last survey that was conducted aimed at evaluating the performance of the application, its usability and the impact of the fidelity level. This last survey was conducted a couple of years after the previous survey, thus with an improved version of the application.

6.5 Future work

Although SketchiXML already provides a wide set of features, several evolutions could be done on the tool itself, new domains of human activity could integrate the technology developed around SketchiXML with some adaptation.

6.5.1 Extending the coverage of sketching artifacts

SketchiXML is supposed to support sketching activities for UIs of information systems involved in multiple contexts of use, the context being understood here as a combination of user, platform, and environment [Calv03]. UI variations have been investigated with respect to user parameters and platform parameters, but not for environmental considerations. Therefore, this last dimension should not be left out.

As such, SketchiXML is focusing GUIs in the development life cycle for multiple contexts. But nothing prevents this tool to sketch

- **GUIs for other types of application than information systems:** provided that the sketched artifacts of a new domain of human activity are captured and expressed either through the grammar or through the gesture recognizer, they could be addressed in principle.
- **Other UI types:** provided that the abstractions required for sketching the artifacts of another UI type (e.g., vocal, tactile, multimodal, virtual reality) are known and represented through the various levels of fidelity, new UI types could be equally sketched.
- **Other UI genres:** the UI type of interest here is mainly a software UI. Physical UIs, or UI mixing different realities, such as mixed reality UIs, tangible UIs, UIs for ambient intelligence, could also be covered in principle.
- **Other models involved in HCI:** although SketchiXML addresses very much the Concrete User Interface (CUI) level by exporting UI specifications at this level in UsiXML terms, it could be imagined that other UsiXML-compliant model (e.g., abstract UI, task, domain, context) could be sketched as well. It would be very nice to benefit from a single design environment where several models involved in the development life cycle could be sketched together, thus leading to the idea of “Sketching it all together”. The models, their internal relationships, and the inter-model relationships.
- **Other model involved outside HCI:** similarly to the observation of extending a sketching tool to encompass a domain model, other non-HCI based models could be subject to sketching activities in the same way. For instance, models involved in UML and RUP are excellent candidates for such an integration. We want for proof that some of these models have already been considered for a sketching tools, but mainly the UML class diagram and the activity diagram. In principle, nothing prevents the sketching tool to consider other types of models, even outside HCI.

6.5.2 Improving the Text Divider

One of the issues that could be better addressed in the future is the text recognition. Indeed, when a designer sketches a future UI, shapes and texts continuously alternate [Hong01]. Thus, the shape recognizer needs to identify whether the stroke received is likely to be a sequence of characters or not. The current approach is quite simple as it merely relies on the number of oscillations with respect to its length and height. We then determine if the stroke was drawn from left to right. Although such an approach provides acceptable results, it has a strong limitation when sketching very short sequence of characters or special characters such as “€” symbol. In order to address this issue correctly, the identification of the text should be contextualized. Other text recognition algorithms should be considered and, perhaps, developed in this context.

6.5.3 Tuning the Recognition Engine more extensively

This thesis presented a new algorithm for multi-stroke gesture recognition based on Levenstein distance. Several improvements could still be carried out on this algorithm in order to improve the performance and more capabilities, such as, but not limited to: bidirectional sketching, sketching with dominant vs. non-dominant hand, multi-hand sketching for a single user or multiple users (e.g., in a group setup). A comparative analysis between the other algorithms for shapes recognition is also worth to conduct, in particular with respect to widely known algorithms such as Rubine [RUBI91]. Recent discussions with experts in that domain lead us to conclude that other algorithms such as Rubine could be tuned effectively so as to cover multi-stroke gestures in the same way our algorithm does. Out of this analysis, a new algorithm could integrate the advantages of the current approaches in a single solution.

6.5.4 Support for Multi-windowing Design

From the case studies involving a web site, many parts of a web page or many fragments of any other UI are likely to come back in other portions of the global UI. The current approach requires the designer to copy and paste all the components that are present on each UI. In order to solve this problem, an easy, simple and usable solution would consist in allowing to the designer to specify if the current sketch must be applied to all UIs (e.g., for multiple platforms or multiple contexts of use) of the project or just to a single one. Thus, changes on the master UI would be reflected on the entire set of UIs. By relying on a

Chapter 6 Conclusion

propagation scheme separating what should be shared in common for all contexts of use and what should remain specific to a particular context. This would support the idea of a genuine “Sketching paradigm for multiple contexts of use”. For this purpose, precise UI transformations should be defined across the contexts of use.

6.5.5 Augmenting the Support for Design Memory

One of the main purposes of SketchiXML consists in easing the communication between the end users and designers. But SketchiXML could be also very useful for conducting collaborative development with a large set of stakeholders. To this end, the tool should be able to keep a trace of all the changes that were made to the project. An interesting improvement would consist in developing a multi-user design memory storing all the information associated to each contributor of the projects over time. In particular, it would be very useful to incorporate some support for design rationale. By attaching a given sketch to comments, it could be possible to record the reasons why such a sketch has been designed and chosen. Multiple benefits could be expected: UI consistency across projects, improved reusability, explicit extension guide, reviewing facilities for any stakeholder, management of design options.

6.5.6 Extending to other domains than Computer Science

As long as sketches are involved in a particular human activity, it could be expected that some support could be provided for this sketching activity, whether this activity is in HCI, in SE, in Computer Science in general or another domain.

To support this statement, the SketchiXML environment has been used in a totally different domain: annotation of mammograms (in the context of the DIAMANT project). Based on the recognition engine and the grammar mechanism developed for our tool, we are currently developing a tool for the annotation of diagnostic images and biomedical signals. These are some domains where sketch-based input would be very useful. This interaction technique enables clinicians to describe their clinical judgment (e.g., anomalies, evaluation, notes) in a very intuitive and effective way. For example in breast cancer screening, the radiologists must carefully report the appearance and oncology classification of the any suspicious lesions. They usually employ a number of descriptors capturing information about the lesion size, its morphology (or clinical type), etc. These descriptors are specific to the type of lesion (Figure 8-2)

Chapter 6 Conclusion

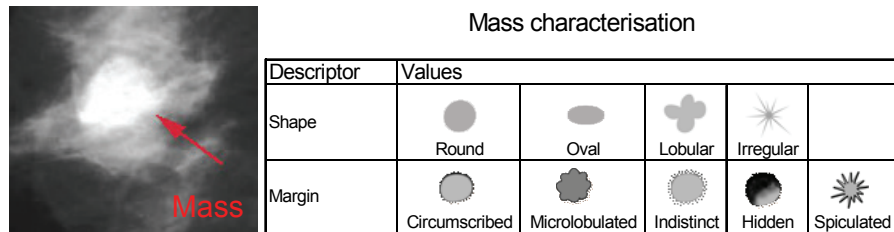


Figure 6-2 Example of features identifying a mass on mammograms

Similarly to the sketching of UIs for rapid-prototyping, the main features of breast cancer lesions can be mapped to a set of graphic symbols defined with the help of users, perhaps with different levels of fidelity. The integration of sketch recognition capabilities into a clinical image annotation system can then provide simple means for reporting and data entry.

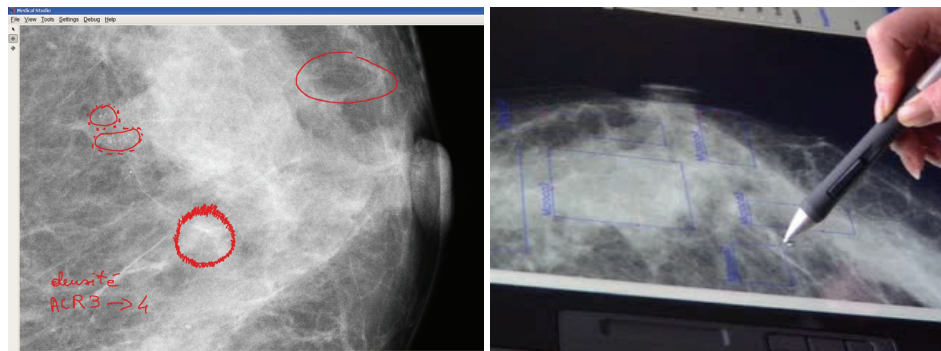


Figure 6-3 Mock-up of sketch based breast cancer annotations

From the user point of view, sketch-based input presents important advantages over a traditional WIMP visualization and annotation tool:

- The annotations can be made directly on the image workspace allowing the radiologist's attention to be focused on the region of interest around the lesion;
- The user can define the set of sketches and symbols according to his personal preferences and does not need to remember complex configurations of the GUIs (menus, control keys, types of functionalities);

Chapter 6 Conclusion

- The interaction paradigm is closer to the manipulation of paper forms currently employed by clinicians and this could be beneficial for users with low skills in computer-based medicine.

Results from an informal user testing have shown pen-based input to be very promising to improve the support of clinical image annotations.

Many other domains could be considered, as long as the objects to be categorized with the application fulfill a specific requirement. Indeed, the world that needs to be modeled using this approach will be defined as a set of interrelated objects. For each kind of object a gesture representation will be associated and relation between these objects will be drawn. Thus, in order to extend the application to other domain, the new domain has to be dividable into objects and relations. For instance, all the examples provided in the first subsection of the future work satisfy this condition.

6.5.7 Extension of UCWorkBench [Ucwo]: a requirements engineering tool

As we presented through the introduction of this thesis, many software engineers tend to consider HCI activities as a part of the application that does not require any specific attention. However, many researches were conducted in order to highlight the importance of such piece of application. Not only because this part of the application was demonstrated as a key component of an application but also because the development of such interface may provide valuable information for the development of the application. Through this thesis we have thus illustrated how to integrate the prototyping framework into the existing methodologies. However, this document has focused on the methodologies as a whole, and never focused on a specific domain of the software engineering: the requirement engineering (RE). This section illustrates how SketchiXML could prove to be a valuable tool for requirements engineering. For this purpose we provide a short description of the requirements engineering and the objectives pursued with such process. The requirements engineering process is typically characterized as an analysis process, where user needs and constraints must be elicited and analyzed. RE is the systematic process of developing requirements through an iterative cooperative process of analyzing a problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained. The essential tasks of RE during software

Chapter 6 Conclusion

engineering (SE) are the elicitation and negotiation of requirements, their specification and validation as well as their management over time [Paec03].

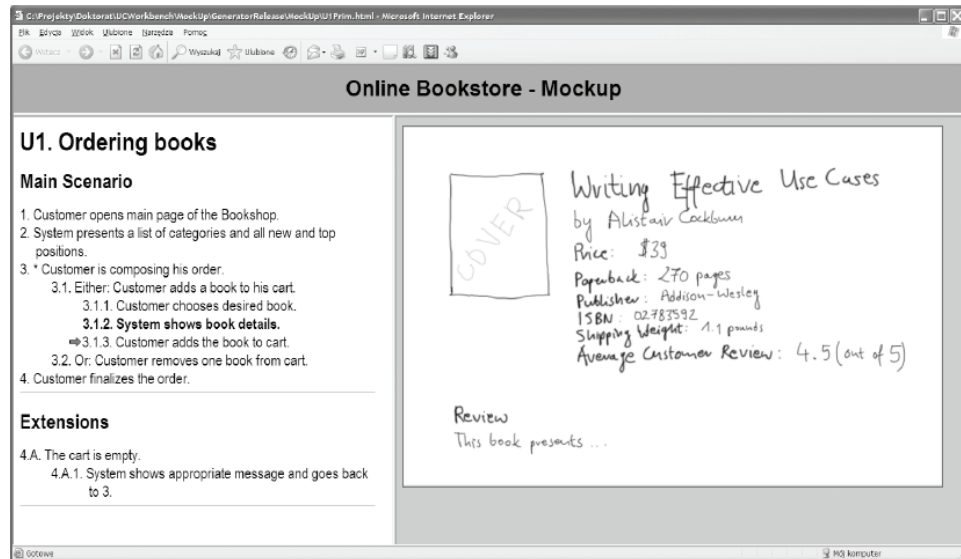


Figure 6-4 Mock-up of sketch based breast cancer annotations

We consider that human interaction should hold an integrated role if software engineering, requirements engineering included. However, human interaction tends to be ignored in RE as many software engineers consider that HCI-considerations can be brought in after the requirements are elicited and that requirements can be elicited without the consideration of the user interface. [Paec03] claims that a fundamental prerequisite for bridging the gap between SE and HCI is that RE is understood as a design activity that includes the design of the user interface. During RE the support for the user through the software system is designed. There are many design decisions to be made such as the to-be-activities of the user tasks supported by the software, the system functions which perform parts of these activities, or the interaction between system and user when the system functions are executed. So the first major observation is that the decision about the tasks is an indispensable prerequisite for starting the RE process. As for example advocated in [Cock01], RE approaches often start with goals. However, there is little guidance on how to identify these goals. Task support is the most important goal, since a system will only be accepted by the users, if their tasks are adequately supported. RE can learn a lot from HCI for the identification of tasks.

Chapter 6 Conclusion

As we stated in this document, SketchiXML should not only be considered as a low fidelity editor, but as a communication mean between designers and end-users. The purpose of the designer in this context consists in sketching the user interface associated to the stories told by the end users. For now, the most common approach for translating such stories into requirements consists in translating them into use cases. Based on the complementarities of use cases and user interface mock up, [Nawr05] developed a new approach to use case engineering. Their approach is based on a combination of a use case editor, a mockup generator and an effort calculator.

The editor uses Formal USE cases (FUSE) language, a simple language formalizing structure of use-cases description. Thus, the use cases definition do not present any particularities itself. The major innovation of the tool consists in linking those uses case with a graphical representation. Based on the definition of the uses cases and the associated collection of user interfaces UCWorkBench generate of mockup of the system.

The mockup generated by the tool consists in a web page displaying all the use cases identified on one side and on the other side their associated representation. Then, the designers and the end-users can validate the use case thanks to an easy navigation.

Considering this approach, we clearly identify an opportunity to develop a new tool based on the advantages of both SketchiXML and UCWorkBench. Indeed, using SketchiXML much more information can be obtained, not the corresponding specifications, but all the information associated to the navigation and the fact that the prototype is really interactive. Moreover, a part of the use cases can be semi-automatically identified based on the specification of the navigation.

6.5.8 General Improvement of sketching facilities

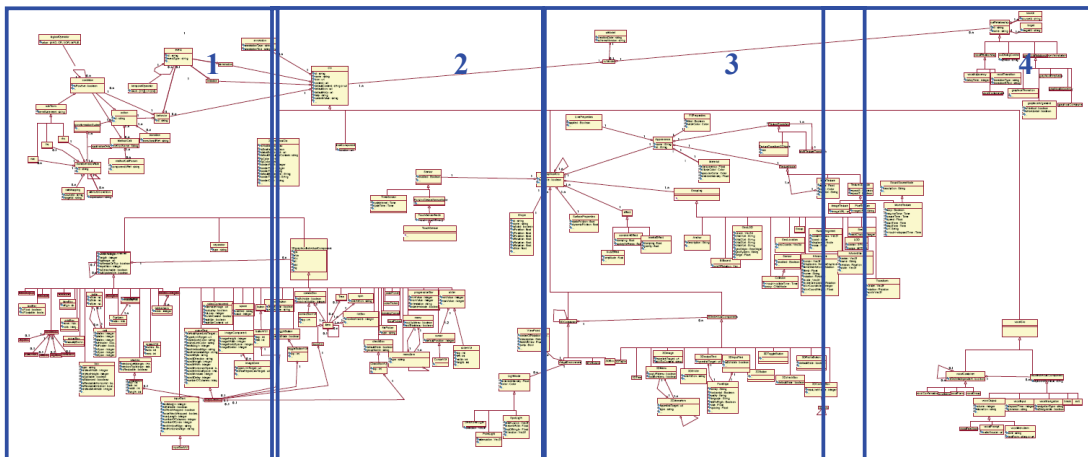
In addition to the set of aforementioned extensions, a lot of work could be devoted to sketching software itself. Indeed, this tool must be considered as an advanced prototype rather than commercially available softwares. Many aspects of the tool could be improved in order to increase its usability and performance. Naturally, the purpose of this thesis was not to develop a commercial product, but to demonstrate the feasibility of such a tool and evaluate its interest in realistic setups. General improvements could consider: implementation optimization for better performance, fine-grained gesture recognition based on fine-tuned parameters, enhanced support for versioning and advanced support for group

Chapter 6 Conclusion

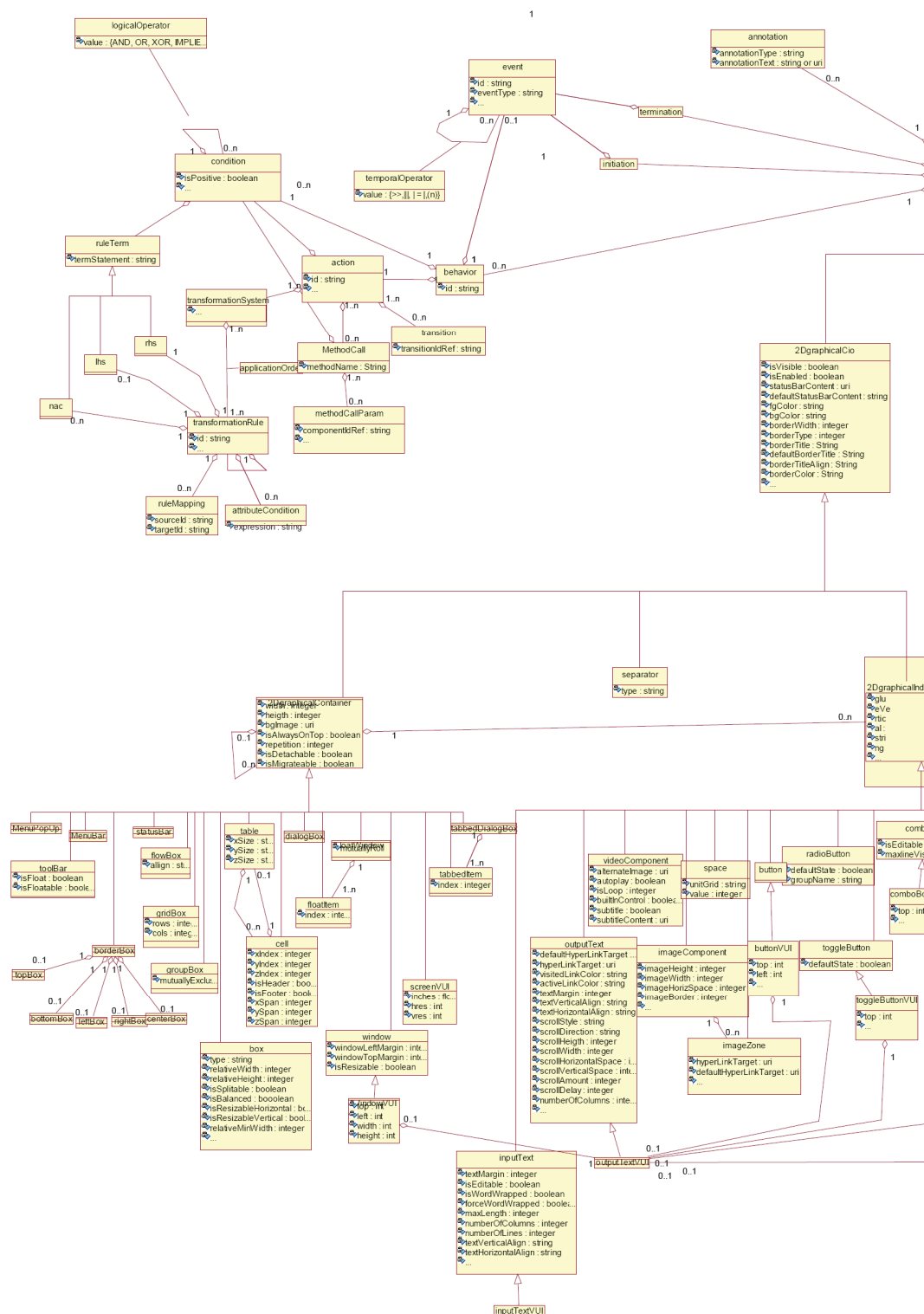
activities. An entirely new series of experiments could be conducted to assess the general usability of a sketching tool like SketchiXML in various industrial setups involving different configurations: a single design, a design with a end user, a end user solely, a team of software designers and developers working together either simultaneously or asynchronously, in a single place or in multiple (remote) places. This represents a new endeavor in the area of group support in development life-cycle.

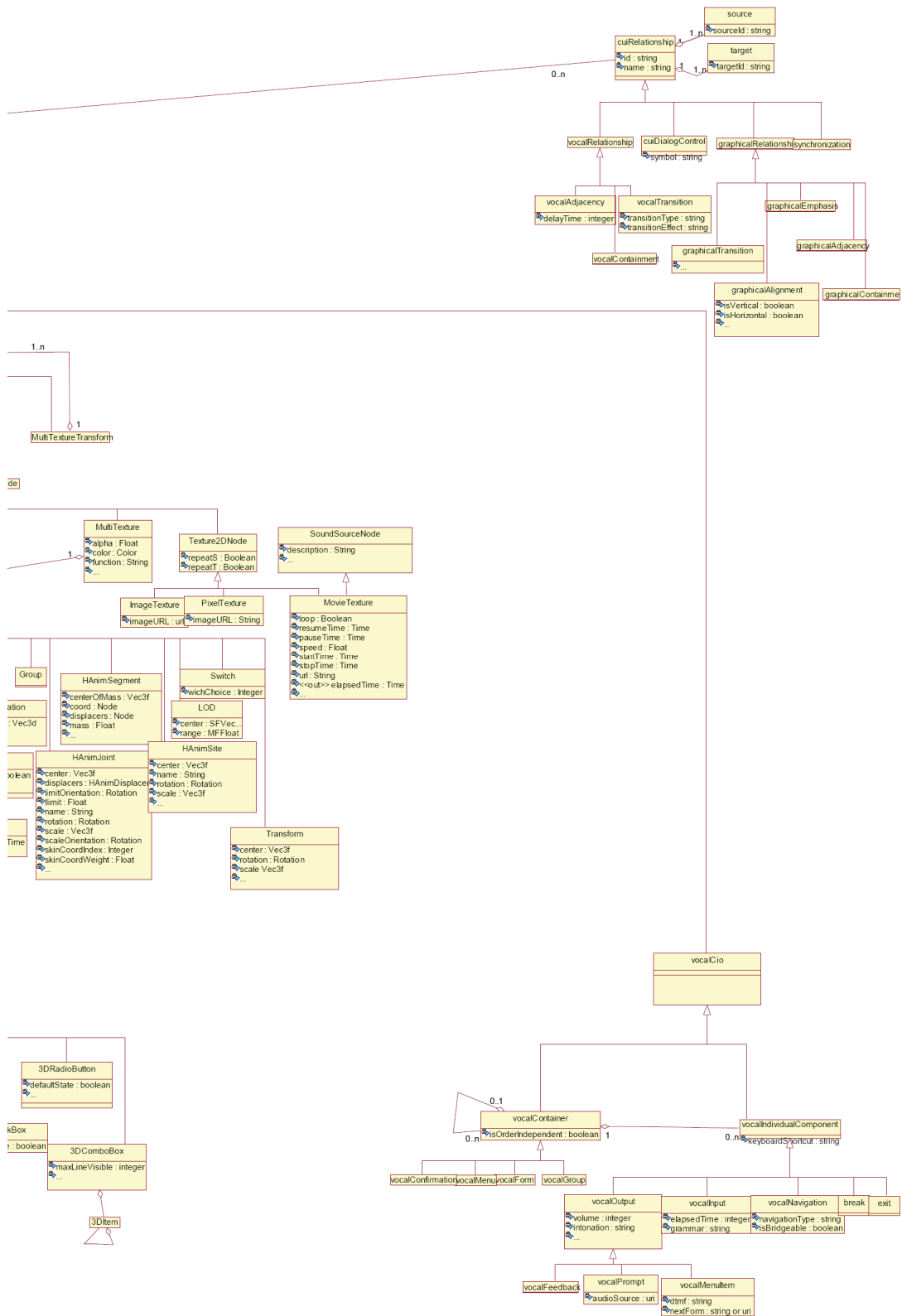
Appendix A - UsiXML 1.8 Class Diagrams

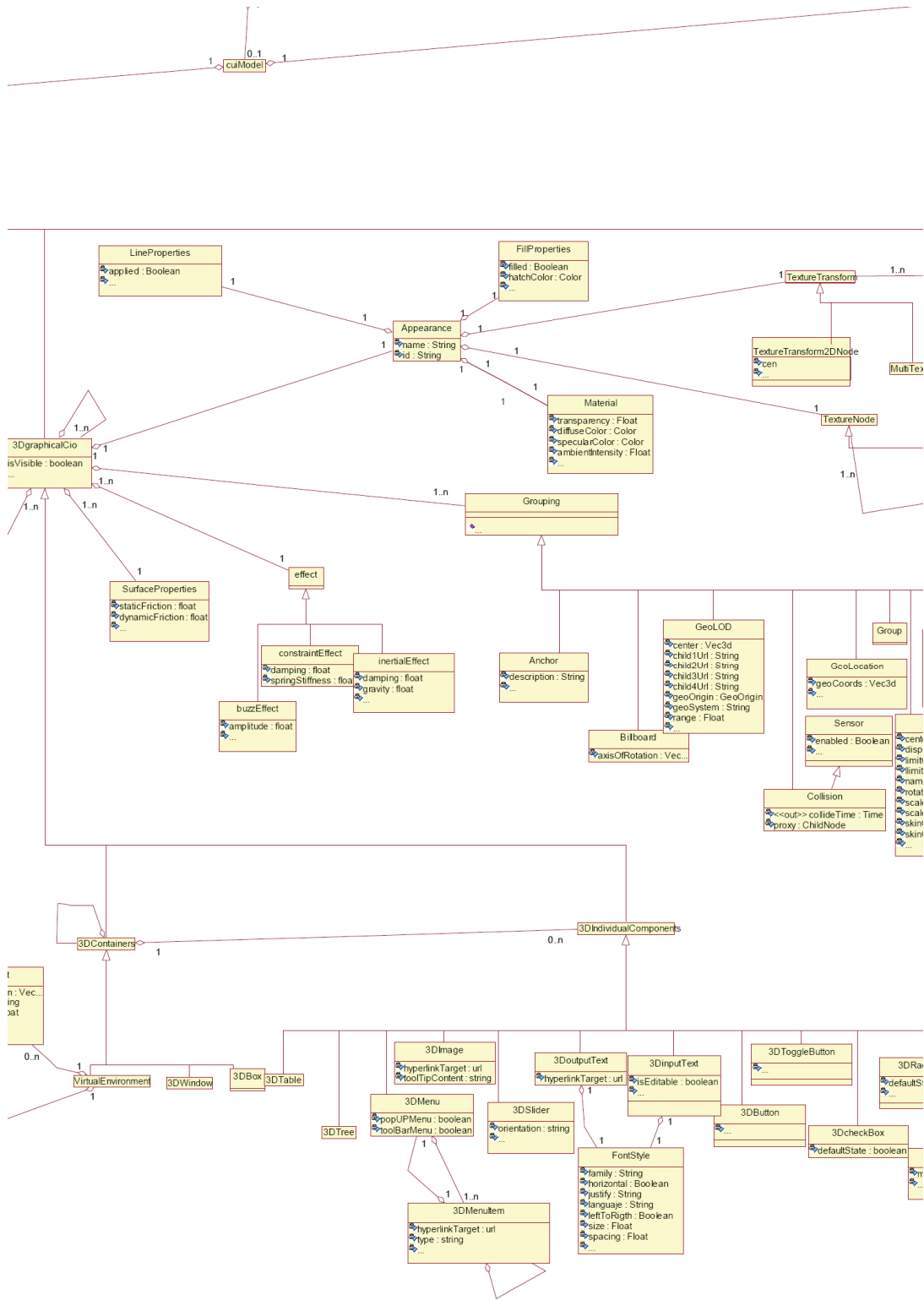
This first appendix contains the meta-model for the UsiXML 1.8 concrete UI model and abstract UI model. Due to the large size of the schema, the CUI meta-model is divided into 4 parts:



Appendix A – UsiXML 1.8 Class Diagram







Appendix B - UsiXML compliant tools

As Many tools based on UsiXML exist, the following section present a selection of tools for each category. Firstly we present the some editor based on UsiXML, then couples of interpreters and eventually other interesting tools.

Editors

Grafixml (<http://www.UsiXML.org/index.php?page=grafixml.xml>), is currently developed at University of Louvain. With this tool, the designer can draw in direct manipulation any graphical UI by directly placing CIOs and editing their properties in a property sheet.

This tool allows users to draw a concrete user interface and then automatically generate UsiXML code from the graphical representation (figure E-2) or to produce an user interface in XHTML, XUL or Java from a UsiXML specification.

Appendix B – UsiXML compliant tools

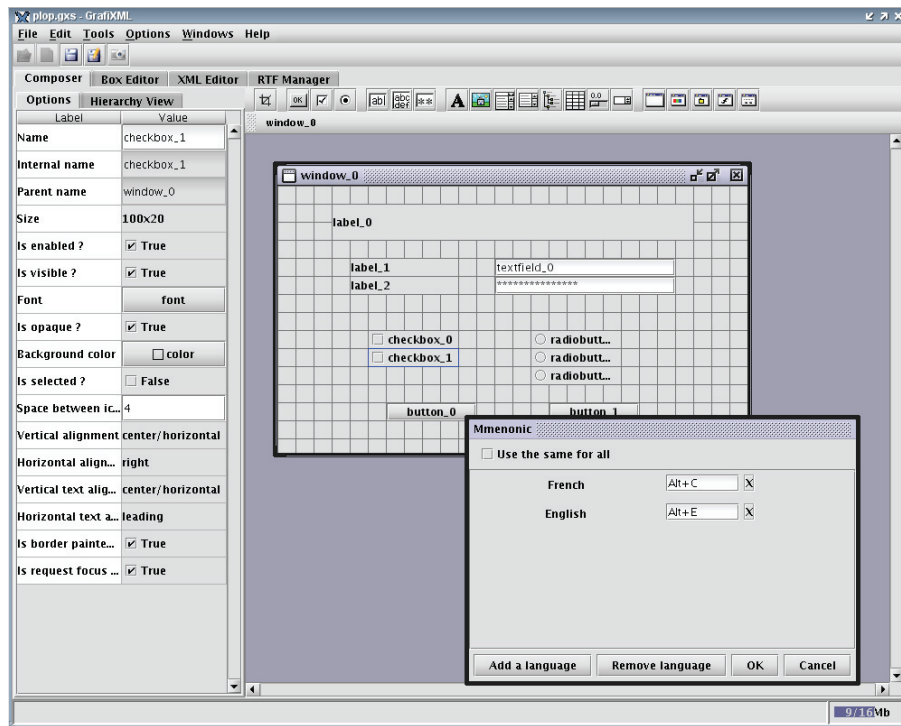
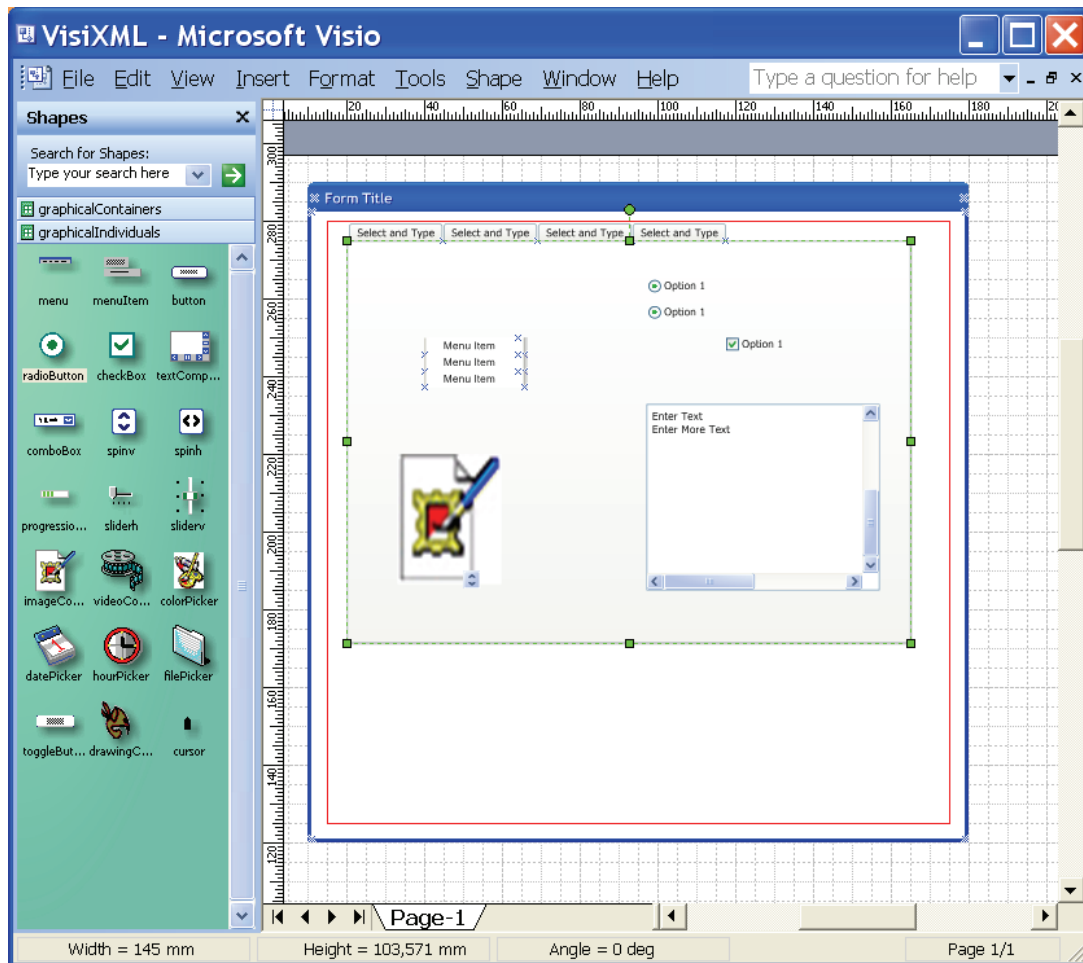


Fig. E-2 GrafiXML

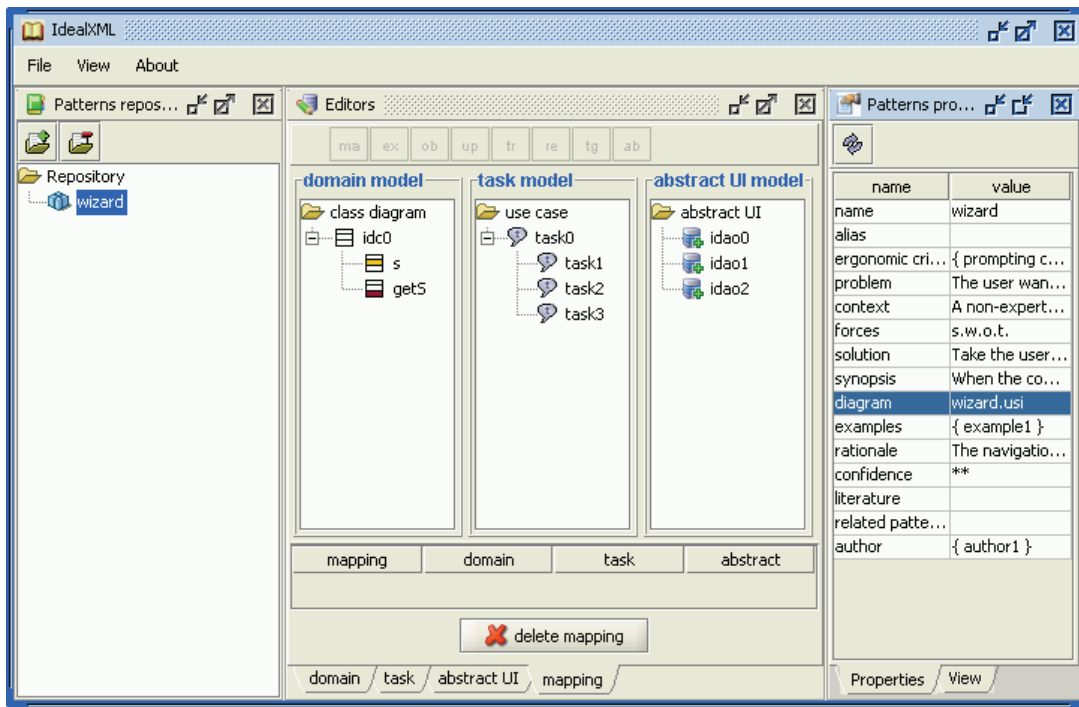
VisiXML (<http://www.usixml.org/index.php?view=page&idpage=11>) is a graphical editor for designing Graphical User Interfaces on top of the Microsoft Visio environment. In this drawing environment, the user drags icons of the language from a palette and drop it onto a working surface area to depict UsiXML elements. VisiXML is intended to support non-developers for mid-fidelity prototyping and specification of Graphical User Interfaces. Therefore, only basic properties of UsiXML elements can be captured to avoid distracting or disturbing the user. Once the design is finished, the tools automatically generates the Abstract User Interface level (AUI) and the Concrete User Interface level (CUI) on demand in UsiXML.

Appendix B – UsiXML compliant tools



IdealXML (Interface Development Environment for AppLications specified in usiXML) is a pattern-oriented tool (<http://www.usixml.org/index.php?view=page&idpage=34>). Using this environment you can edit, manipulate, view and learn about patterns. It is based on Pattern Language Markup Language (PLML), but additional elements were provided. You can create a new repository, and then distribute your repository to other peoples, this is essential. You can edit textual features associated with a pattern, such as: name, alias, problem, context, solution, synopsis, rationale, etc. And you can edit diagrams using meaningful notations (class diagrams and CTT) from software engineering and human-computer interaction. So, ergonomic criteria, forces, diagrams, examples and author's information are associated with a pattern too. Diagrams are associated using UsiXML and patterns are stored using PLML.

Appendix B – UsiXML compliant tools



Teresa is developed at ISTI-CNR and supports the generation of XHTML, VoiceXML and WML starting from a task model, an abstract or concrete UI model expressed in TeresaXML or a concrete UI specified in UsiXML.

Interpreters

FlashiXML (<http://www.usixml.org/index.php?view=page&idpage=15>) is a rendering engine of UsiXML-compliant UIs in a vectorial mode that is SVG-compatible. Any UsiXML-compliant UI can be opened and rendered in this interpreter so as to create the truly working UIs with presentation and dialog. In this environment, the UI can be resized at any time to address some constraints imposed by the computing platforms and to support some properties of Graceful Degradation of UIs, a sub-property of the Plasticity property. In this way, any UsiXML-compliant UI can be rendered on any computing platform equipped with a SVG or Flash plug-in.

Appendix B – UsiXML compliant tools

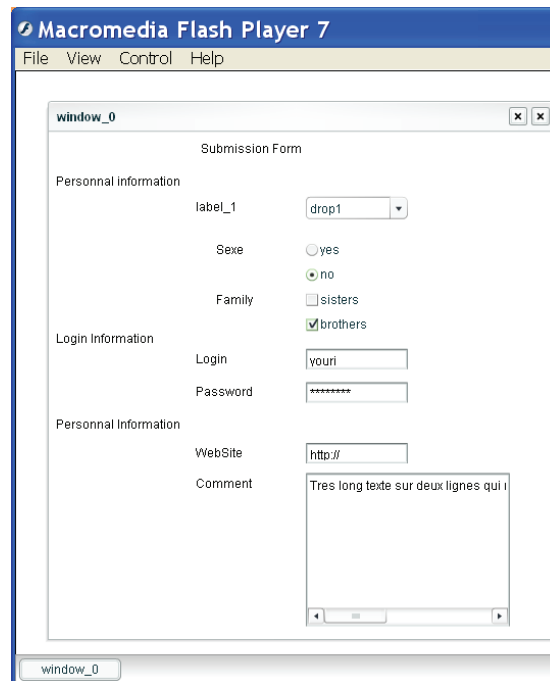


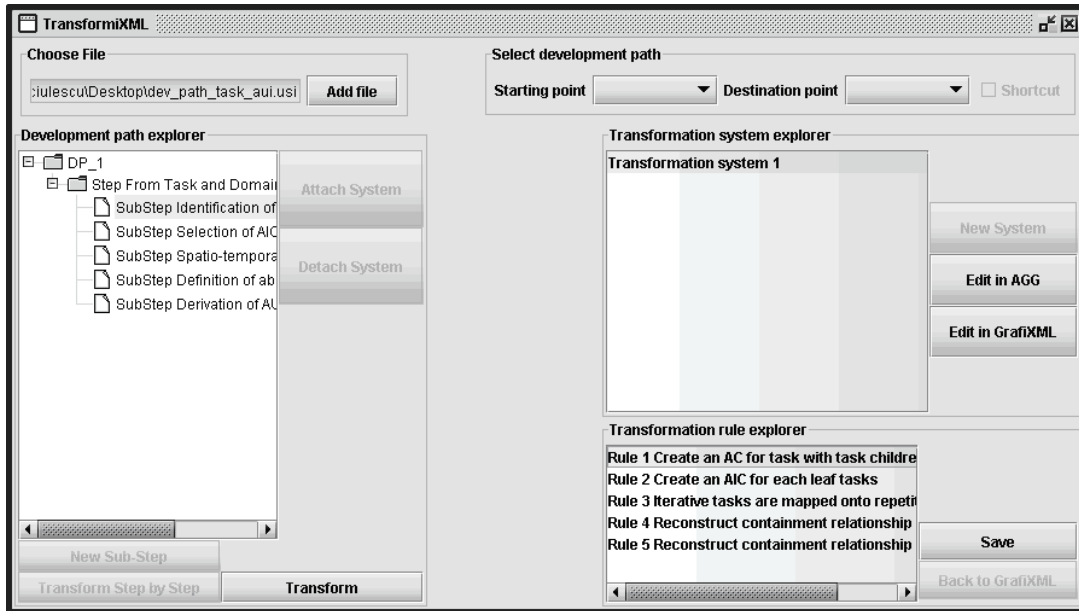
Fig E-3 FlashiXML

InterpiXML (<http://www.usixml.org/index.php?view=page&idpage=41>) is a runtime interpreter for UsiXML files. In GrafiXML, you are able to automatically generate a Java description of a user interface, but in this project, it is expected that the end user will receive a series of UsiXML files containing interfaces for different tasks, so as to make her/his "To do" list. In this way, the end user can "open" one or several interfaces and execute them instantly to mimic the principle of the unique workstation. This project is currently under initial development. Another facility provided by InterpiXML is its capability to change the native language of the user and re-launch the user interface and/or to change the presentation look& feel dynamically.

Other tools

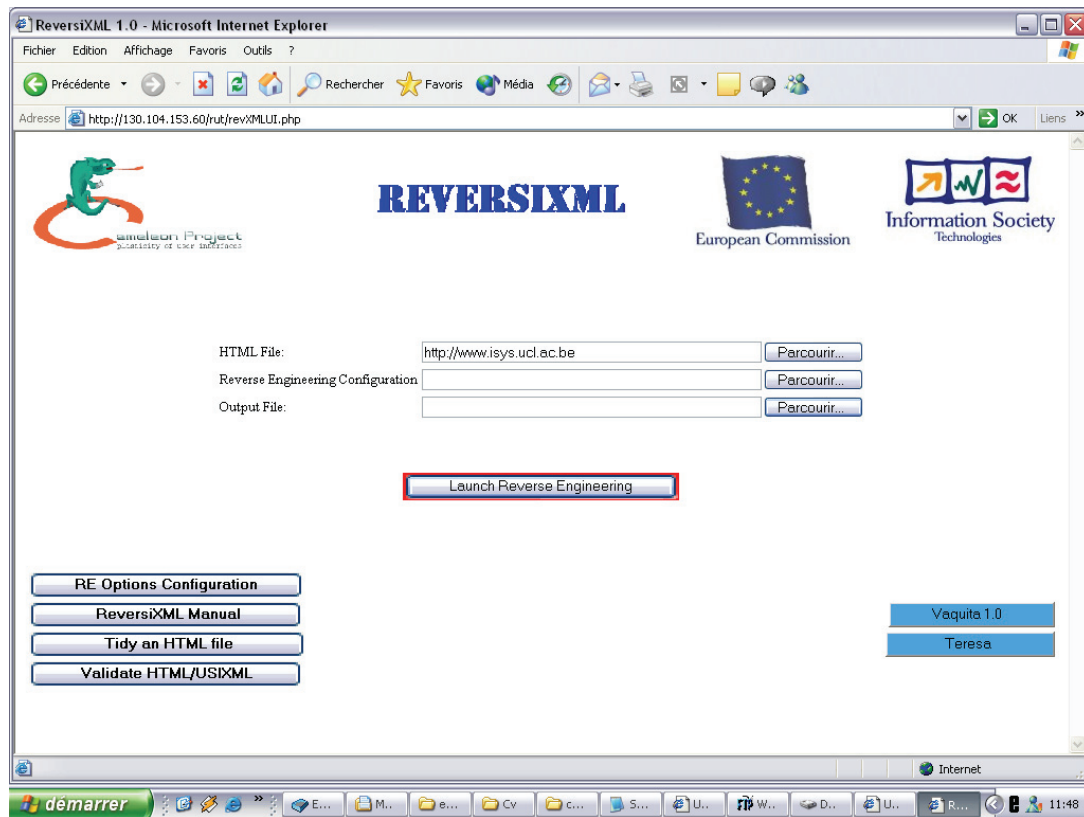
TransformiXML, applies graph transformations contained in graph grammars to perform transformations of UsiXML-compliant UIs to produce a new UI specification. Such transformation can occur between any level (task and domain, abstract user interface, concrete user interface) to support forward engineering, reverse engineering, middle-out approach, adaptation, and the wide spreading approach. The tool allows managing a development library (a library containing a catalog of transformation rules)

Appendix B – UsiXML compliant tools



ReversiXML is a tool that reverse engineers any HTML page of a Web site into UsiXML, both at Abstract User Interface (AUI) and the Concrete User Interface (CUI) levels so as to retarget an existing web site to another computing platform. ReversiXML is the online version of Vaquita which is a set of techniques established to reverse engineer UIs that were not designed according to a model-based approach. In this manner, these UIs can be incorporated in the same pipe-line and this allow migration of UIs from one computing platform to another. The current goal of ReversiXML is to reverse engineer a web site onto a concrete or abstract UI model according to flexible heuristics. The flexibility of this process is of high importance considering the many design options that may have been decided at design time.

Appendix B – UsiXML compliant tools



MigriXML is a virtual reality system representing the user's real environment, based on UsiXML models: the platforms found in that environment, the UI of interactive graphics applications that are executed on these platforms, and the user. Within that virtual environment, the user interacts with the platforms and the running applications as if they were their real counterparts.

The main characteristic of MigriXML is that it supports the run-time user interface migration between computing platforms. The user can select any application, and make the related UI emigrate from the source platform and immigrate in a target platform. To do so, the user presses the button (M) –which stands for 'migrate'– that can be found in the button bar of the application window. As a result, the user 'grabs' the window, and from that very moment the window will follow the user's cursor within the screen of the source platform and any other platform, being rendered according to the resolution and definition of the pointed screen.

UsabilityAdviser is a tool intended to improve usability and accessibility of user interfaces designed with an usixml editor. It interacts with the usixml editor to find violated usability and accessibility rules during the design of an user interface.

Appendix B – UsiXML compliant tools

UsabilityAdviser is based on separating the ergonomic knowledge from the evaluation engine. Indeed, the knowledge base of usability rules is a simple text file which contains the description of rules in a formal language. This formal language is very similar to the natural language. This separation provide a dynamic and flexible structuring of this knowledge according to the rapid evolution technologies and scientific findings in the fields of ergonomics and human factors.

For the moment, UsabilityAdviser is only compatible with SketchiXml and GrafiXml. But it can easily be adapted to other tools thanks to the standard of communication which was developed.









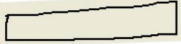


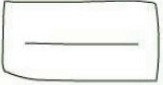





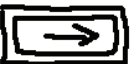
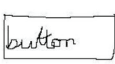



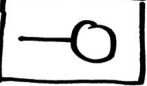

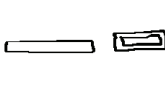
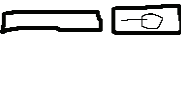


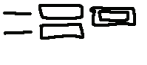


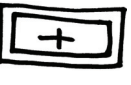
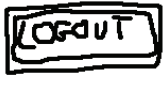

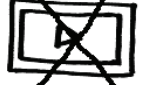




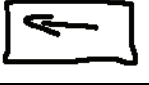



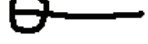

The screenshot shows the UsabilityAdviser application window with a menu bar (File, View, Rule, Tools, Help) and a toolbar. The main area displays a table of usability rules. The table has columns for Id, Certainty, Cover, Category, Priority, Description, and Enabled.

| Id | Certainty | Cover | Category | Priority | Description | Enabled |
|----|-----------|---------|--------------------------|----------|--|-------------------------------------|
| 1 | high | partial | Visibilité et perception | must | Le blanc et le jaune sont des paires de couleurs à ne p... | <input checked="" type="checkbox"/> |
| 2 | medium | partial | Perception du contenu | should | Eviter de mettre des phrases dans les libellés des ém... | <input checked="" type="checkbox"/> |
| 3 | high | partial | Perception du contenu | should | Fournir une alternative textuelle pour chacune des ém... | <input checked="" type="checkbox"/> |
| 4 | high | partial | Perception du contenu | must | Fournir une alternative textuelle pour chacune des ém... | <input checked="" type="checkbox"/> |
| 5 | medium | partial | Perception du contenu | should | Limiter le nombre d'images de décoration à 4 | <input checked="" type="checkbox"/> |
| 6 | high | total | Surcharge | should | Limiter le nombre d'icônes des toolbars à 20 | <input checked="" type="checkbox"/> |
| 7 | high | total | Surcharge | may | Limiter le nombre d'icônes des toolbars à 12 | <input checked="" type="checkbox"/> |
| 8 | high | total | Perception du contenu | should | Chaque combobox possède 5 à 9 items | <input checked="" type="checkbox"/> |
| 9 | medium | partial | Navigation | should | Les boutons de contrôle doivent être situés en bas de ... | <input checked="" type="checkbox"/> |
| 10 | medium | partial | Navigation | should | Les boutons de contrôle doivent être alignés sur le mé... | <input checked="" type="checkbox"/> |
| 11 | high | partial | Perception du contenu | must | Utiliser des polices de caractères standards | <input checked="" type="checkbox"/> |
| 12 | high | total | Perception du contenu | must | Chaque zone d'image comporte un lien redondant | <input checked="" type="checkbox"/> |
| 13 | high | total | Perception du contenu | must | Chaque élément multimédia comporte une alternative ... | <input checked="" type="checkbox"/> |
| 14 | high | total | Perception du contenu | must | Chaque zone d'image comporte un lien redondant | <input checked="" type="checkbox"/> |



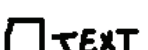
















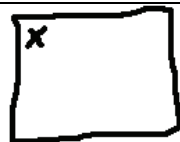
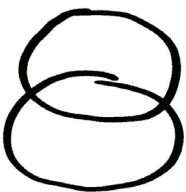
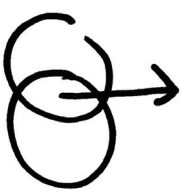









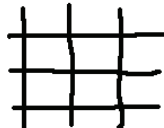
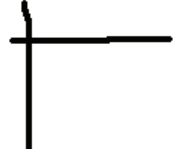







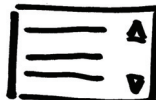

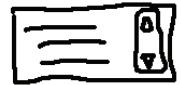


Appendix B – UsiXML compliant tools

Appendix C – Widgets catalogue






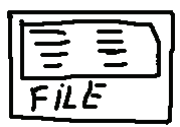

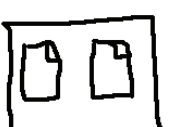




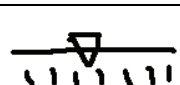
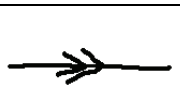
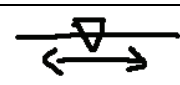
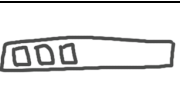
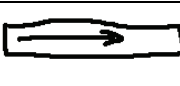
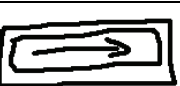
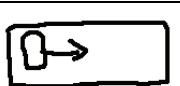
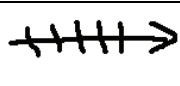



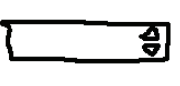
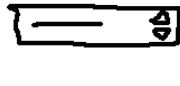
The first table of this chapter contains all the representations that were submitted to the end users and designers during the survey for the widgets catalogue construction.

| | | | | | |
|--------------|---|---|--|---|---|
| Text |  |  |  |  | <i>Requiere la formulação para o texto Please, complete the form and use</i> |
| Textfield |  |  |  |  |  |
| Text Area |  |  |  |  |  |
| Button |  |  |  |  |  |
| Search Field |  |  |  |  |  |
| Login |  |  |  |  | Login |
| Logout |  |  |  |  |  |
| Reset |  |  |  |  |  |
| Radio Button |  |  |  |  |  |




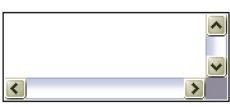





Appendix C – Building a Widget Catalogue

| | | | | | |
|-----------------|---|---|--|---|---|
| Check Box |  |  |  |  |  |
| Validate |  |  |  |  |  |
| Combobox |  |  |  |  |  |
| Image |  |  |  |  |  |
| Multimedia area |  |  |  |  |  |
| Group Box |  |  |  |  |  |
| Table |  |  |  |  |  |
| Hyperlink |  |  |  |  |  |
| Listbox |  |  |  |  |  |

Appendix C – Building a Widget Catalogue

| | | | | | |
|---------------|---|---|--|---|---|
| Toggle Button |  |  |  |  |  |
| File Picker |  |  |  |  |  |
| Slider |  |  |  |  |  |
| Progress Bar |  |  |  |  |  |
| Spinner |  |  |  |  |  |







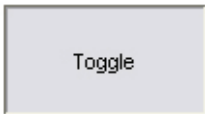





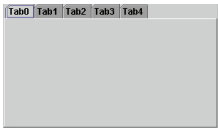

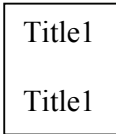

The following table shows all the widgets to be handled by SketchiXML. This list is far from being exhaustive as the grammar can be enriched easily.

| Widget Type | Graphical Representation | Sketching propositions |
|--------------|---|--|
| Text | This is text |  |
| TextField |  |  |
| TextArea |  |  |
| Button |  |  |
| Search Field |  |  |

Appendix C – Building a Widget Catalogue

| | | | | | | | | | | | | | | | | | |
|------------------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
| Login | <div>User name <input type="text"/></div> <div>Password <input type="password"/></div> <div>Log in</div> | | | | | | | | | | | | | | | | |
| Log out | <div>Log out</div> | | | | | | | | | | | | | | | | |
| Reset Form | <div>Reset</div> | | | | | | | | | | | | | | | | |
| Validate | <div>Validate</div> | | | | | | | | | | | | | | | | |
| RadioButton | <div><input checked="" type="radio"/> Radio Button</div> | | | | | | | | | | | | | | | | |
| CheckBox | <div><input type="checkbox"/> Check Box</div> | | | | | | | | | | | | | | | | |
| Combobox | <div><input type="text"/></div> | | | | | | | | | | | | | | | | |
| Image | <div>Image</div> | | | | | | | | | | | | | | | | |
| Multi Media Area | <div>Multi-media</div> | | | | | | | | | | | | | | | | |
| Layer | <div><div></div><div></div></div> | | | | | | | | | | | | | | | | |
| Group Box | <div>Group Box</div> | | | | | | | | | | | | | | | | |
| Table | <div><table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table></div> | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | |
| Hyperlink | <div>Hyperlink</div> | | | | | | | | | | | | | | | | |

Appendix C – Building a Widget Catalogue

| | | |
|---------------|---|--|
| Anchor |  |  |
| ListBox |  |  |
| Hour Picker |  |  |
| Toggle Button |  |  |
| Slider |  |  |
| Progress Bar |  |  |
| TabDialogBox |  |  |
| Menu |  |  |

Appendix C – Building a Widget Catalogue

Appendix D – UsiXML specification

This section contains the complete specification generated by SketchiXML for the second case study.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <uiModel id="Case_study_2" name="Case_study_2" creationDate="2007-04-
03T16:16:58.109+02:00" schemaVersion="1.6.4" xmlns="http://www.usixml.org">
- <head>
  <version modifDate="2007-04-03T16:16:58.109+02:00" />
  <authorName>Adrian</authorName>
  <comment>This file was generated with SketchiXML</comment>
  <comment>Information on this tool can be found on www.usixml.org</comment>
</head>
- <cuiModel id="Case_study_2-cui" name="Case_study_2-cui">
- <window id="window_0" name="window_0" isVisible="true" isEnabled="true" width="240"
height="320" isAlwaysOnTop="false" isResizable="true">
- <gridBagBox id="Box_0" name="Box_0" gridHeight="16" gridWidth="12">
- <constraint gridx="2" gridy="0" gridwidth="5" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_0" name="Label_0" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="7" gridy="1" gridwidth="2" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <imageComponent id="Picture_0" name="Picture_0" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" textColor="#000000" />
</constraint>
- <constraint gridx="0" gridy="3" gridwidth="8" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_1" name="Label_1" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="0" gridy="6" gridwidth="3" gridheight="9" weightx="1.0" weighty="1.0"
fill="none">
  <imageComponent id="Picture_1" name="Picture_1" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" textColor="#000000" />
</constraint>
```

Appendix D – Case Study (UsiXML Specification)

```
-<constraint gridx="4" gridy="7" gridwidth="6" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Hyperlink_1" name="Hyperlink_1" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false"
isSubscript="false" isSuperscript="false" isPreformatted="false" textColor="#000000"
textFont="Dialog" isItalic="false" defaultHyperLinkTarget="http://www.usixml.org"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="4" gridy="9" gridwidth="8" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Hyperlink_2" name="Hyperlink_2" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false"
isSubscript="false" isSuperscript="false" isPreformatted="false" textColor="#000000"
textFont="Dialog" isItalic="false" defaultHyperLinkTarget="http://www.usixml.org"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="4" gridy="11" gridwidth="6" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Hyperlink_0" name="Hyperlink_0" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false"
isSubscript="false" isSuperscript="false" isPreformatted="false" textColor="#000000"
textFont="Dialog" isItalic="false" defaultHyperLinkTarget="http://www.usixml.org"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="6" gridy="14" gridwidth="6" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Hyperlink_3" name="Hyperlink_3" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false"
isSubscript="false" isSuperscript="false" isPreformatted="false" textColor="#000000"
textFont="Dialog" isItalic="false" defaultHyperLinkTarget="http://www.usixml.org"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
</gridBagBox>
</window>
- <window id="window_1" name="window_1" isVisible="true" isEnabled="true" width="240"
height="320" isAlwaysOnTop="false" isResizable="true">
- <gridBagBox id="Box_0" name="Box_0" gridHeight="16" gridWidth="12">
- <constraint gridx="2" gridy="0" gridwidth="5" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_0" name="Label_0" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="1" gridy="3" gridwidth="8" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_1" name="Label_1" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="8" gridy="1" gridwidth="2" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
```

Appendix D – Case Study (UsiXML Specification)

```
<imageComponent id="Picture_0" name="Picture_0" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" textColor="#000000" />
</constraint>
- <constraint gridx="0" gridy="5" gridwidth="3" gridheight="9" weightx="1.0" weighty="1.0"
fill="none">
  <imageComponent id="Picture_1" name="Picture_1" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" textColor="#000000" />
  </constraint>
- <constraint gridx="5" gridy="5" gridwidth="4" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_2" name="Label_2" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
  </constraint>
- <constraint gridx="4" gridy="7" gridwidth="1" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_3" name="Label_3" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
  </constraint>
- <constraint gridx="6" gridy="7" gridwidth="4" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
  <comboBox id="ComboBox_0" name="ComboBox_0" isVisible="true" isEnabled="true"
textColor="#000000" />
  </constraint>
- <constraint gridx="4" gridy="9" gridwidth="2" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_4" name="Label_4" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
  </constraint>
- <constraint gridx="6" gridy="9" gridwidth="4" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
  <inputText id="TextField_0" name="TextField_0" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ffffff" textColor="#000000" maxLength="100"
numberOfColumns="20" numberOfLines="1" isPassword="false" isWordWrapped="true"
forceWordWrapped="true" isEditable="true" defaultFilter="" />
  </constraint>
- <constraint gridx="4" gridy="11" gridwidth="2" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_5" name="Label_5" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
  </constraint>
- <constraint gridx="6" gridy="11" gridwidth="4" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
  <comboBox id="ComboBox_1" name="ComboBox_1" isVisible="true" isEnabled="true"
textColor="#000000" />
  </constraint>
```

Appendix D – Case Study (UsiXML Specification)

```
-<constraint gridx="6" gridy="14" gridwidth="5" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Hyperlink_0" name="Hyperlink_0" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false"
isSubscript="false" isSuperscript="false" isPreformatted="false" textColor="#000000"
textFont="Dialog" isItalic="false" defaultHyperLinkTarget="http://www.usixml.org"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
</gridBagBox>
</window>
-<window id="window_2" name="window_2" isVisible="true" isEnabled="true" width="240"
height="320" isAlwaysOnTop="false" isResizable="true">
-<gridBagBox id="Box_0" name="Box_0" gridHeight="16" gridWidth="12">
-<constraint gridx="2" gridy="0" gridwidth="5" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_0" name="Label_0" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
-<constraint gridx="1" gridy="3" gridwidth="8" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_1" name="Label_1" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
-<constraint gridx="8" gridy="1" gridwidth="2" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <imageComponent id="Picture_0" name="Picture_0" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" textColor="#000000" />
</constraint>
-<constraint gridx="0" gridy="6" gridwidth="3" gridheight="9" weightx="1.0" weighty="1.0"
fill="none">
  <imageComponent id="Picture_1" name="Picture_1" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" textColor="#000000" />
</constraint>
-<constraint gridx="4" gridy="7" gridwidth="7" gridheight="4" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_2" name="Label_2" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
-<constraint gridx="4" gridy="11" gridwidth="5" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Hyperlink_0" name="Hyperlink_0" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false"
isSubscript="false" isSuperscript="false" isPreformatted="false" textColor="#000000"
textFont="Dialog" isItalic="false" defaultHyperLinkTarget="http://www.usixml.org"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
-<constraint gridx="6" gridy="14" gridwidth="5" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
```


Appendix D – Case Study (UsiXML Specification)

```
<outputText id="Hyperlink_1" name="Hyperlink_1" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false"
isSubscript="false" isSuperscript="false" isPreformatted="false" textColor="#000000"
textFont="Dialog" isItalic="false" defaultHyperLinkTarget="http://www.usixml.org"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
</gridBagBox>
</window>
- <window id="window_3" name="window_3" isVisible="true" isEnabled="true" width="240"
height="320" isAlwaysOnTop="false" isResizable="true">
- <gridBagBox id="Box_0" name="Box_0" gridHeight="16" gridWidth="12">
- <constraint gridx="2" gridy="0" gridwidth="5" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
<outputText id="Label_0" name="Label_0" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="1" gridy="2" gridwidth="8" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
<outputText id="Label_1" name="Label_1" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="8" gridy="0" gridwidth="2" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
<imageComponent id="Picture_0" name="Picture_0" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" textColor="#000000" />
</constraint>
- <constraint gridx="0" gridy="5" gridwidth="3" gridheight="9" weightx="1.0" weighty="1.0"
fill="none">
<imageComponent id="Picture_1" name="Picture_1" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" textColor="#000000" />
</constraint>
- <constraint gridx="4" gridy="5" gridwidth="3" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
<outputText id="Label_2" name="Label_2" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="4" gridy="7" gridwidth="8" gridheight="7" weightx="1.0" weighty="1.0"
fill="none">
<listBox id="ListBox_0" name="ListBox_0" isVisible="true" isEnabled="true"
textColor="#000000" />
</constraint>
- <constraint gridx="7" gridy="15" gridwidth="5" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
<outputText id="Hyperlink_0" name="Hyperlink_0" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false"
isSubscript="false" isSuperscript="false" isPreformatted="false" textColor="#000000"
textFont="Dialog" isItalic="false" defaultHyperLinkTarget="http://www.usixml.org"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
```

Appendix D – Case Study (UsiXML Specification)

```
</constraint>
</gridBagBox>
</window>
- <window id="window_4" name="window_4" isVisible="true" isEnabled="true" width="240"
height="320" isAlwaysOnTop="false" isResizable="true">
- <gridBagBox id="Box_0" name="Box_0" gridHeight="16" gridWidth="12">
- <constraint gridx="2" gridy="0" gridwidth="5" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_0" name="Label_0" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
  </constraint>
- <constraint gridx="8" gridy="1" gridwidth="2" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <imageComponent id="Picture_0" name="Picture_0" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" textColor="#000000" />
  </constraint>
- <constraint gridx="0" gridy="5" gridwidth="3" gridheight="9" weightx="1.0" weighty="1.0"
fill="none">
  <imageComponent id="Picture_1" name="Picture_1" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" textColor="#000000" />
  </constraint>
- <constraint gridx="0" gridy="3" gridwidth="3" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_1" name="Label_1" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
  </constraint>
- <constraint gridx="4" gridy="3" gridwidth="2" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Hyperlink_0" name="Hyperlink_0" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false"
isSubscript="false" isSuperscript="false" isPreformatted="false" textColor="#000000"
textFont="Dialog" isItalic="false" defaultHyperLinkTarget="http://www.usixml.org"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
  </constraint>
- <constraint gridx="6" gridy="14" gridwidth="4" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Hyperlink_1" name="Hyperlink_1" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false"
isSubscript="false" isSuperscript="false" isPreformatted="false" textColor="#000000"
textFont="Dialog" isItalic="false" defaultHyperLinkTarget="http://www.usixml.org"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
  </constraint>
- <constraint gridx="4" gridy="5" gridwidth="2" gridheight="4" weightx="1.0" weighty="1.0"
fill="none">
  <imageComponent id="Picture_2" name="Picture_2" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" textColor="#000000" />
  </constraint>
- <constraint gridx="7" gridy="4" gridwidth="3" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
```

Appendix D – Case Study (UsiXML Specification)

```
<outputText id="Label_2" name="Label_2" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="7" gridy="6" gridwidth="3" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
<outputText id="Label_3" name="Label_3" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="4" gridy="10" gridwidth="6" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
<outputText id="Label_4" name="Label_4" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="4" gridy="12" gridwidth="6" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
<outputText id="Label_5" name="Label_5" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="4" gridy="13" gridwidth="6" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
<outputText id="Label_6" name="Label_6" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="4" gridy="14" gridwidth="2" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
<outputText id="Hyperlink_2" name="Hyperlink_2" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false"
isSubscript="false" isSuperscript="false" isPreformatted="false" textColor="#000000"
textFont="Dialog" isItalic="false" defaultHyperLinkTarget="http://www.usixml.org"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
</gridBagBox>
</window>
- <window id="window_5" name="window_5" isVisible="true" isEnabled="true" width="240"
height="320" isAlwaysOnTop="false" isResizable="true">
- <gridBagBox id="Box_0" name="Box_0" gridHeight="16" gridWidth="12">
- <constraint gridx="2" gridy="0" gridwidth="5" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
<outputText id="Label_0" name="Label_0" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
```

Appendix D – Case Study (UsiXML Specification)

```
- <constraint gridx="1" gridy="3" gridwidth="3" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_1" name="Label_1" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="8" gridy="1" gridwidth="2" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <imageComponent id="Picture_0" name="Picture_0" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" textColor="#000000" />
</constraint>
- <constraint gridx="4" gridy="3" gridwidth="2" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Hyperlink_0" name="Hyperlink_0" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false"
isSubscript="false" isSuperscript="false" isPreformatted="false" textColor="#000000"
textFont="Dialog" isItalic="false" defaultHyperLinkTarget="http://www.usixml.org"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="0" gridy="5" gridwidth="2" gridheight="8" weightx="1.0" weighty="1.0"
fill="none">
  <imageComponent id="Picture_1" name="Picture_1" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" textColor="#000000" />
</constraint>
- <constraint gridx="4" gridy="4" gridwidth="6" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_2" name="Label_2" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="4" gridy="6" gridwidth="2" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_3" name="Label_3" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="3" gridy="7" gridwidth="4" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_4" name="Label_4" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
- <constraint gridx="3" gridy="8" gridwidth="2" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_5" name="Label_5" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
```

Appendix D – Case Study (UsiXML Specification)

```
-<constraint gridx="3" gridy="9" gridwidth="3" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_6" name="Label_6" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
-<constraint gridx="3" gridy="11" gridwidth="3" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_7" name="Label_7" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
-<constraint gridx="7" gridy="6" gridwidth="4" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
  <inputText id="TextField_0" name="TextField_0" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ffffff" textColor="#000000" maxLength="100"
numberOfColumns="20" numberOfLines="1" isPassword="false" isWordWrapped="true"
forceWordWrapped="true" isEditable="true" defaultFilter="" />
</constraint>
-<constraint gridx="7" gridy="7" gridwidth="3" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
  <inputText id="TextField_1" name="TextField_1" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ffffff" textColor="#000000" maxLength="100"
numberOfColumns="20" numberOfLines="1" isPassword="false" isWordWrapped="true"
forceWordWrapped="true" isEditable="true" defaultFilter="" />
</constraint>
-<constraint gridx="7" gridy="8" gridwidth="4" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
  <inputText id="TextField_2" name="TextField_2" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ffffff" textColor="#000000" maxLength="100"
numberOfColumns="20" numberOfLines="1" isPassword="false" isWordWrapped="true"
forceWordWrapped="true" isEditable="true" defaultFilter="" />
</constraint>
-<constraint gridx="7" gridy="9" gridwidth="4" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
  <inputText id="TextField_3" name="TextField_3" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ffffff" textColor="#000000" maxLength="100"
numberOfColumns="20" numberOfLines="1" isPassword="false" isWordWrapped="true"
forceWordWrapped="true" isEditable="true" defaultFilter="" />
</constraint>
-<constraint gridx="7" gridy="11" gridwidth="4" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
  <inputText id="TextField_4" name="TextField_4" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ffffff" textColor="#000000" maxLength="100"
numberOfColumns="20" numberOfLines="1" isPassword="false" isWordWrapped="true"
forceWordWrapped="true" isEditable="true" defaultFilter="" />
</constraint>
</gridBagBox>
</window>
-<window id="window_6" name="window_6" isVisible="true" isEnabled="true" width="240"
height="319" isAlwaysOnTop="false" isResizable="true">
-<gridBagBox id="Box_0" name="Box_0" gridHeight="0" gridWidth="0">
```

Appendix D – Case Study (UsiXML Specification)

```
-<constraint gridx="2" gridy="0" gridwidth="5" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_0" name="Label_0" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
-<constraint gridx="8" gridy="0" gridwidth="2" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <imageComponent id="Picture_0" name="Picture_0" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" textColor="#000000" />
</constraint>
-<constraint gridx="0" gridy="5" gridwidth="3" gridheight="9" weightx="1.0" weighty="1.0"
fill="none">
  <imageComponent id="Picture_1" name="Picture_1" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" textColor="#000000" />
</constraint>
-<constraint gridx="3" gridy="6" gridwidth="8" gridheight="7" weightx="1.0" weighty="1.0"
fill="none">
  <listBox id="ListBox_0" name="ListBox_0" isVisible="true" isEnabled="true"
textColor="#000000" />
</constraint>
-<constraint gridx="7" gridy="14" gridwidth="4" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Hyperlink_0" name="Hyperlink_0" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false"
isSubscript="false" isSuperscript="false" isPreformatted="false" textColor="#000000"
textFont="Dialog" isItalic="false" defaultHyperLinkTarget="http://www.usixml.org"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
-<constraint gridx="0" gridy="2" gridwidth="3" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_1" name="Label_1" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
-<constraint gridx="4" gridy="2" gridwidth="3" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Hyperlink_1" name="Hyperlink_1" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false"
isSubscript="false" isSuperscript="false" isPreformatted="false" textColor="#000000"
textFont="Dialog" isItalic="false" defaultHyperLinkTarget="http://www.usixml.org"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
-<constraint gridx="4" gridy="5" gridwidth="6" gridheight="1" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_2" name="Label_2" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
</constraint>
</gridBagBox>
</window>
```


Appendix D – Case Study (UsiXML Specification)

```
- <window id="window_7" name="window_7" isVisible="true" isEnabled="true" width="240"
height="319" isAlwaysOnTop="false" isResizable="true">
- <gridBagBox id="Box_0" name="Box_0" gridHeight="0" gridWidth="0">
- <constraint gridx="2" gridy="0" gridwidth="5" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_0" name="Label_0" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
  </constraint>
- <constraint gridx="0" gridy="2" gridwidth="3" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_2" name="Label_2" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
  </constraint>
- <constraint gridx="7" gridy="0" gridwidth="2" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <imageComponent id="Picture_0" name="Picture_0" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" textColor="#000000" />
  </constraint>
- <constraint gridx="0" gridy="5" gridwidth="3" gridheight="9" weightx="1.0" weighty="1.0"
fill="none">
  <imageComponent id="Picture_1" name="Picture_1" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" textColor="#000000" />
  </constraint>
- <constraint gridx="3" gridy="7" gridwidth="8" gridheight="7" weightx="1.0" weighty="1.0"
fill="none">
  <listBox id="ListBox_0" name="ListBox_0" isVisible="true" isEnabled="true"
textColor="#000000" />
  </constraint>
- <constraint gridx="7" gridy="14" gridwidth="4" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Hyperlink_0" name="Hyperlink_0" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false"
isSubscript="false" isSuperscript="false" isPreformatted="false" textColor="#000000"
textFont="Dialog" isItalic="false" defaultHyperLinkTarget="http://www.usixml.org"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
  </constraint>
- <constraint gridx="4" gridy="3" gridwidth="3" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Hyperlink_1" name="Hyperlink_1" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false"
isSubscript="false" isSuperscript="false" isPreformatted="false" textColor="#000000"
textFont="Dialog" isItalic="false" defaultHyperLinkTarget="http://www.usixml.org"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
  </constraint>
- <constraint gridx="4" gridy="4" gridwidth="5" gridheight="2" weightx="1.0" weighty="1.0"
fill="none">
  <outputText id="Label_1" name="Label_1" isVisible="true" isEnabled="true" fgColor="#000000"
bgColor="#ece9d8" isBold="true" isUnderline="false" isStrikethrough="false" isSubscript="false"
isSuperscript="false" isPreformatted="false" textColor="#000000" textFont="Dialog" isItalic="false"
visitedLinkColor="#000000" textMargin="0" numberOfColumns="15" numberOfLines="1" />
```

Appendix D – Case Study (UsiXML Specification)

```
</constraint>
</gridBagBox>
</window>
</cuiModel>
- <contextModel id="Case_study_2-contextModel_0" name="Case_study_2-contextModel">
- <context id="Case_study_2-contextModel_0" name="Case_study_2-context-en_US">
  <userStereotype id="Case_study_2-sten_US_9" language="en_US"
stereotypeName="Case_study_2-sten_US" />
  <platform id="Case_study_2-platform_0" name="Case_study_2-platform" />
  <environment id="Case_study_2-env_0" name="Case_study_2-env" />
</context>
</contextModel>
- <resourceModel>
- <cioRef cioId="Label_0">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_1">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Hyperlink_1">
  <resource content="Hyperlink" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Hyperlink_2">
  <resource content="Hyperlink" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Hyperlink_0">
  <resource content="Hyperlink" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Hyperlink_3">
  <resource content="Hyperlink" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_0">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_1">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_2">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_3">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_4">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_5">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Hyperlink_0">
  <resource content="Hyperlink" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Button_0">
  <resource content="Button" contextId="Case_study_2-contextModel_0" />
```


Appendix D – Case Study (UsiXML Specification)

```
</cioRef>
- <cioRef cioId="Label_0">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_1">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_2">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Hyperlink_0">
  <resource content="Hyperlink" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Hyperlink_1">
  <resource content="Hyperlink" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_0">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_1">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_2">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Button_0">
  <resource content="Button" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Hyperlink_0">
  <resource content="Hyperlink" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_0">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_1">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Hyperlink_0">
  <resource content="Hyperlink" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Hyperlink_1">
  <resource content="Hyperlink" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_2">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_3">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_4">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_5">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
```

Appendix D – Case Study (UsiXML Specification)

```
</cioRef>
- <cioRef cioId="Label_6">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Hyperlink_2">
  <resource content="Hyperlink" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_0">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_1">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Hyperlink_0">
  <resource content="Hyperlink" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_2">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_3">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_4">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_5">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_6">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_7">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Button_0">
  <resource content="Button" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_0">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Hyperlink_0">
  <resource content="Hyperlink" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_1">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Hyperlink_1">
  <resource content="Hyperlink" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_2">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_0">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
```

Appendix D – Case Study (UsiXML Specification)

```
</cioRef>
- <cioRef cioId="Label_2">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Hyperlink_0">
  <resource content="Hyperlink" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Hyperlink_1">
  <resource content="Hyperlink" contextId="Case_study_2-contextModel_0" />
</cioRef>
- <cioRef cioId="Label_1">
  <resource content="Label" contextId="Case_study_2-contextModel_0" />
</cioRef>
</resourceModel>
</uiModel>
```

Appendix D – Case Study (UsiXML Specification)

Appendix E – SketchiXML User Guide



User Guide



The Belgian Laboratory of
Computer-Human Interaction
Adrien Coyette

Appendix E SketchiXML User Guide

Table of content

| | | |
|-----|--|----|
| 1. | System requirements | 3 |
| 2. | Starting the application | 3 |
| 3. | Parameterize the application | 4 |
| 4. | Elements of the SketchiXML Environment | 5 |
| 5. | Interacting with SketchiXML | 6 |
| 6. | Building widgets | 8 |
| 7. | Editing functions | 9 |
| 8. | Gesture training | 9 |
| 9. | Text recognition | 11 |
| 10. | Grammar edition | 11 |
| 11. | Level of fidelity | 13 |
| 12. | Widgets catalogue | 14 |
| 13. | General Information | 17 |
| 14. | Acknowledgements | 17 |

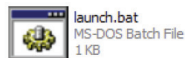
Appendix E SketchiXML User Guide

1. System requirements

- Minimal System Specifications
 - ✓ Java 2 version 1.5
 - ✓ A Windows operating system
 - ✓ At least 256 MB RAM
 - ✓ A mouse
- Recommended System Specifications
 - ✓ Java 2 version 1.5
 - ✓ A Windows XP operating system
 - ✓ At least 512 MB RAM
 - ✓ A mouse
 - ✓ A pen tablet that can emulate a mouse
 - ✓ If the pen tablet your are using is compatible with the wintab driver, install the driver so as to enable the advanced pen capabilities
 - ✓ Download and install the Microsoft Windows XP Tablet PC Edition Software Development Kit 1.7 (in order to use the text recognition)

2. Starting the application

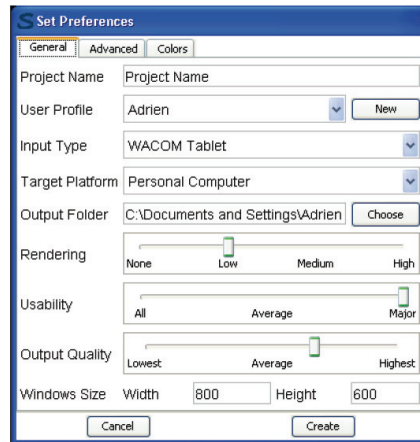
- ✓ Download SketchiXML and unzip the archive
- ✓ All the required libraries are included in the package
- ✓ Double-click on the “launch.bat” file to start the application



- ✓ If an error message appears at startup, ignore this, this just mean that no wintab drivers were found (driver for tablet pc, graphics tablet...).

3. Parameterize the application

When SketchiXML starts, a dialog box is displayed asking the user to provide a set of parameters for the application.



This set of parameters will drive the low-fidelity prototyping process:

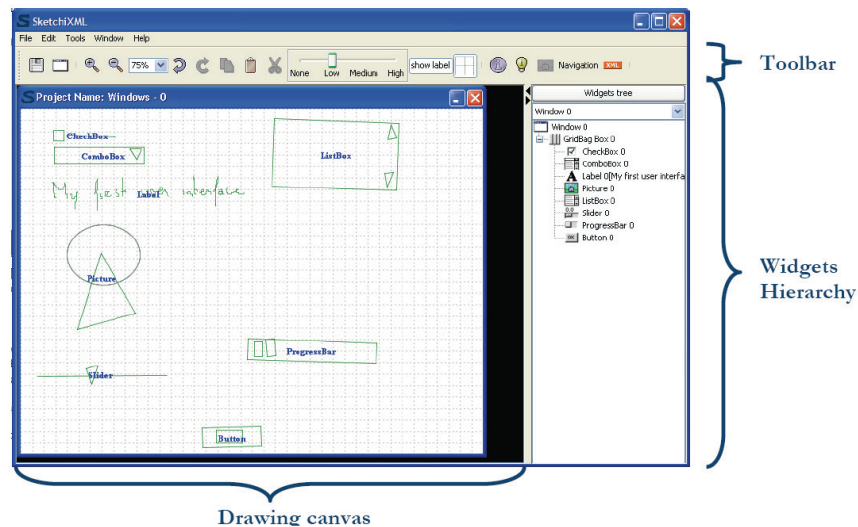
- ✓ The project name: determines the name of the current project
- ✓ The user profile: each user of the system will have a different configuration and will train the system according to his/ her preferences.
- ✓ The input type: determines what kind of input device to use. The selection list displays all the compatible devices found on the computer
- ✓ The target platform: determines for which computing platform the UI is prototyped (the actual version of SketchiXML only offers two choices with personal computer and pda, but other platform can be added easily).
- ✓ The output folder: allows to choose the location to save the files.
- ✓ The rendering level: this slider let the user choose the level of rendering to be used by the application. As it is presented later in the document, the user can choose between four different levels of fidelity ranging from nothing to the high level.

Appendix E SketchiXML User Guide

- ✓ The usability level: this slider allows to define how SketchiXML is supposed to interact with the Usability adviser. The usability adviser is a third party application providing real time advice on the design process.
- ✓ The output quality: the value selected on this slider will affect the way the sketches are interpreted. When estimating the spatial constraints between the shapes, the result will depend of this value. A lower value increases the speed while a higher value gives a more reliable result.
- ✓ The windows size: this two text fields let the user specify the initial size of the windows to be created.

All the value provided here can be changer during the design process, the next sections will illustrates how to change it.

4. Elements of the SketchiXML Environment



The SketchiXML design window is divided into four parts

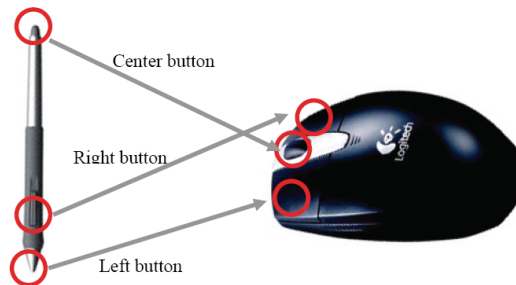
- ✓ The drawing canvas is the area where you can sketch your user interfaces
- ✓ The widgets tree on the right of the screen shows the hierarchy of all the recognized widgets.

Appendix E SketchiXML User Guide

- ✓ The toolbar provide access to all the frequent operation such as copy, paste, cut...
- ✓ The menu bar provide access to all the functions including the functions present in the tool bar

5. Interacting with SketchiXML

- ✓ There are different possible means of interaction with SketchiXML, you can either use the application by using a standard mouse or by using a pen based device. If your pen-enabled device support custom configuration, you should use this feature and configure the pen as displayed here under. If your pen-enabled device is configured as described bellow, you must choose the mouse as input for SketchiXML.



In both case, three different actions can be done

- ✓ You can **draw** a new widget or part of a widget. If you are using a pen enabled device you just have to draw on the drawing surface. If you are using a mouse you should draw on the drawing area while keeping the right button pressed.



Appendix E SketchiXML User Guide

- ✓ You can **delete** a widget or a part of a widget. If you are using a pen enabled device you just need to use the eraser present on the back of the pen. If you are using a mouse you should use the center button (most of the time the wheel). Then, draw a stroke on the widgets / shapes you want to remove.



- ✓ You can call a **command** using the left button on the mouse or by using the button present of the pen if you are using a pen enabled device.

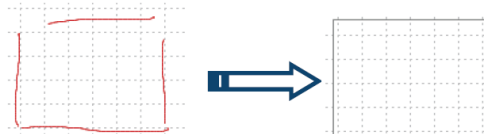


6. Building widgets

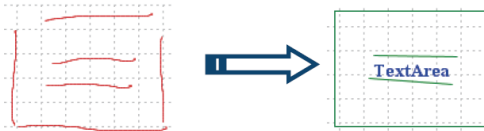
In order to build widget, you need to use the drawing function. Each widget is made up of a construct of one to several shapes / gesture / widgets.

- ✓ Since the shapes can be multi-stroke, it is compulsory to observe a delay of half a second (default value) between the strokes, otherwise the recognitions engine will not recognize the sketch properly. This delay can be changed on the advanced tab of the dialog box showed at the beginning of the process.

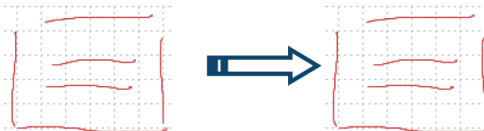
A multi-strokes rectangle where each stroke is considered to be part of the same shape (delay smaller than 0.5 second between strokes)



A text area based on the rectangle where a delay of half a second was respected between the drawing of the rectangle and both lines

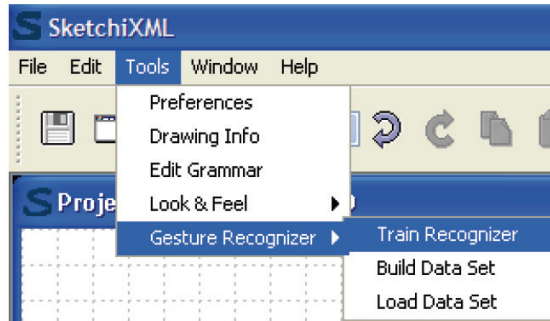


A text area based on the rectangle where delay of half a second was not respected between the drawing of the rectangle and both lines

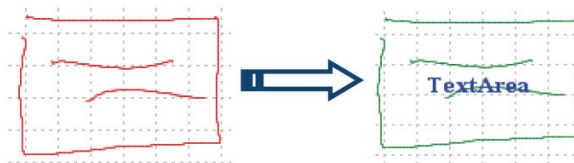


- ✓ For each widget one or several representations can be associated. These representations are made of a set of shapes / widgets / gestures and a set of constraints to be checked amongst the component to be used for the construction.
- ✓ For instance, one of the representations for the text area is composed of three shapes and four constraints. The three shapes are the two lines and the rectangle. The constraints require that the two lines are horizontal and are present inside of the rectangle. Many other representations can be added easily thanks to the grammar edition interface (presented in 10).

Appendix E SketchiXML User Guide



- ✓ The training interface lets you define a set of new representation for shapes, widgets and commands. You can even add new types of shapes and widgets by clicking the add menu. The new shape you will create here will be available in the grammar editor and can be defined to be part of a widget. The new widget you declare here will not have multiple representations for the different levels of fidelity.
- ✓ When using the gestures, you are not bound to the half second delay restriction between the strokes. For instance this gesture was not recognized previously, but once a gesture was defined for this widget, it's recognized correctly.



- ✓ Contrary to the shape primitive recognition engine, the gesture recognizer is not very flexible. It means that you have to redraw the gesture in a very similar manner that you did when defining the pattern. As an example, if you draw a horizontal line from right to left, it's completely different from a line from left to right. This is the reason why we use profile in SketchiXML, each user has to define its own set of gestures.

7. Editing functions

- ✓ SketchiXML supports all the standard editing functions. All these functions can be accessed from the toolbar, the edit menu and by using commands.



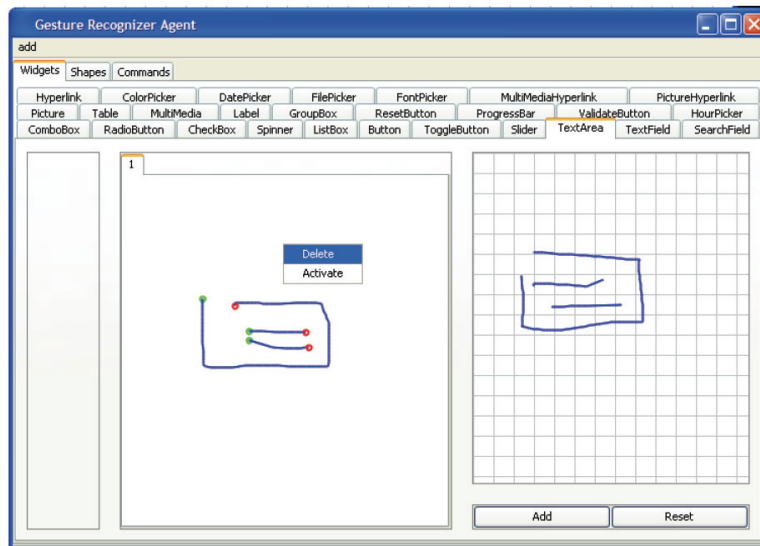
- ✓ The designer is able to choose the zooming level by either using the magnifier icons or the combo box.
- ✓ The undo and redo functions let the designer going back to a previous stage of development. There are no limitations on the number of action to be stored in the undo list.
- ✓ The copy, paste and cut operations require that a part of the user interface was selected (cut and copy) or copied (paste)
- ✓ In order to select a part of the user interface, the designer has to surround the area to be selected with a dashed line (at least 5 tokens)



8. Gesture training

- ✓ If you do not train the system with new gesture, SketchiXML will only be able to recognize a set of shape primitives: circle, ellipse, triangle, rectangle, line, cross... This is sufficient to build most of the simple widget, but in case you want to build complex widget such as a file picker, you may need some extra shapes. To this end, SketchiXML propose a gesture training interface.

Appendix E SketchiXML User Guide



- ✓ You can specify several gestures for a same widget, shape or command even if it is not recommended. The gesture recognition process is far from being light, so a large number of gestures may affect the performance of the application.

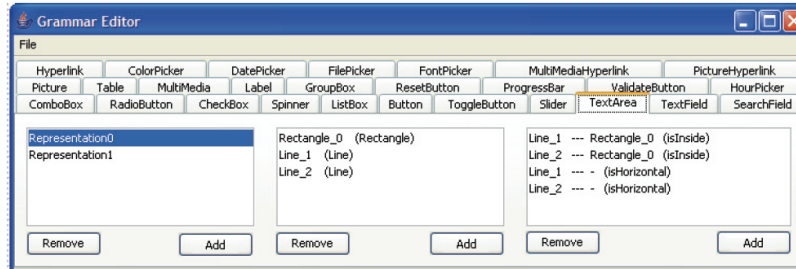
9. Text recognition

- ✓ SketchiXML is supporting handwriting recognition. Since SketchiXML is using the recognition engine provided with Tablet PC, make sure you are either using a tablet pc, or install the Microsoft Windows XP Tablet PC Edition Software Development Kit 1.7. This development kit is freely downloadable from the Microsoft website.

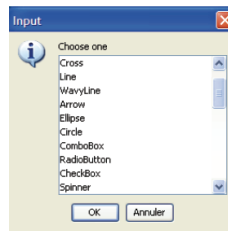
10. Grammar edition

- ✓ The grammar edition panel allows to specify new representation for the widgets. For each widget several representations can be added. Each of these representations consists in a group of shapes, widgets or gesture and a list of constraints.

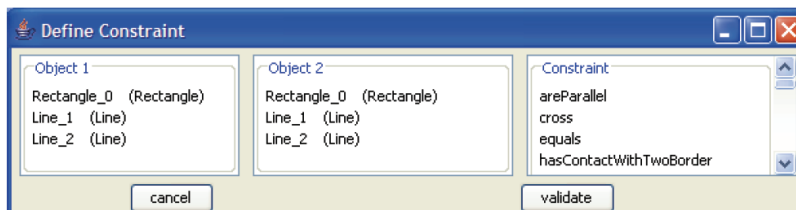
Appendix E SketchiXML User Guide



- ✓ In order to add a new representation you have to click on the “add” button under the representation list. Then, you need to choose the different shape to be used in the representation. This selection can be done by clicking on the “add” button under the shape list. A list of all the available shape is then displayed.



- ✓ We can observe that the list contains a set of shape primitives, and also all the different types of widget and all the different types of gestures.
- ✓ Each time a new shape is added to the shape list, the shape receives a unique name. Once the list of all the shapes to be used in the representation is defined, we can specify the list of constraints. To this aim we have to specify constraint on each individual shape or pair or shapes. To add a constraint, click on the “add” button located under the constraints list, a dialog box will be displayed

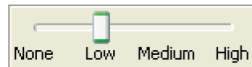


Appendix E SketchiXML User Guide

- ✓ To specify a constraint, select a shape in the “Object 1” list. If the constraint you want to specify does not need a second shape you can directly specify the constraints, otherwise you need to select a second shape in the “Object 2” list. When the constraints is ready, click on the validate button and the constraint is added to the list of constraint associated with this representation.

11. Level of fidelity

- ✓ SketchiXML lets the designer choose the rendering level. This level can be chosen at the beginning of the process, but can be changer at any stage of development freely.



- ✓ The “**none**” level corresponds to a situation where nothing is displayed on the screen. The designer does not get any feed back on his drawing.

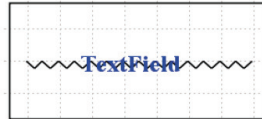


- ✓ The “**low**” level corresponds to a situation all the shapes and widgets that are recognized are displayed on the screen. A shape that is recognized is displayed in grey and replaced by its equivalent, and a widget that is recognized is displayed in green with a label in the centre.



- ✓ The “**medium**” level corresponds to a situation where all the widget that are recognized are replaced by a smoother informal representation of this widget. For the recognized shapes, there are not any changes with the low level of fidelity.

Appendix E SketchiXML User Guide



- ✓ The “**high**” level corresponds to a situation where all the widgets that are recognized are replaced by their corresponding widget in the java Swing toolkit. For the recognized shapes, there are not any changes with the low level of fidelity.












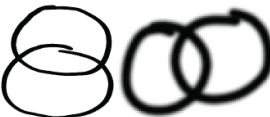




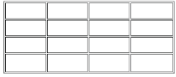




TextField_0

12. Widgets catalogue






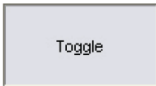






- ✓ The following section presents all the widgets that are supported in SketchiXML. For most of them we have predefined a set of representations, but this is not rigid. You can add new representations and remove the existing ones.

| Widget Type | Graphical Representation | Sketching proposition |
|--------------|--------------------------|------------------------------|
| Text | This is text | |
| TextField | | |
| TextArea | | |
| Button | | |
| Search Field | | No predefined representation |
| Login | | No predefined representation |
| Log out | | No predefined representation |
| Reset Form | | No predefined representation |
| Validate | | No predefined representation |

Appendix E SketchiXML User Guide

| | | |
|----------------------------|--|---|
| Radio Button |  Radio Button |  |
| Checkbox |  Check Box |  |
| Combo box |  |  |
| Image |  |  |
| Image hyperlink |  |  |
| Multi Media Area |  |  |
| Multi Media Area hyperlink |  |  |
| Group Box |  |  |
| Table |  |  |
| Hyperlink | Hyperlink |  |
| ListBox |  |  |

Appendix E SketchiXML User Guide

| | | |
|----------------------|---|---|
| Menu |  | No predefined representation |
| Color Picker |  | No predefined representation |
| File Picker |  | No predefined representation |
| Date Picker |  | No predefined representation |
| Hour Picker |  | No predefined representation |
| Toggle Button |  |  |
| Slider |  |  |
| Progress Bar |  |  |
| Spinner |  | No predefined representation |

13. General Information

- ✓ This tool was developed by Adrien Coyette, member of Belgian Laboratory of Computer Human Interaction (www.isys.ucl.ac.be/bchi/)
- ✓ Address
Adrien Coyette
Place des doyens, 1 office A110
1348 Louvain-la-Neuve
Belgium
Coyette@isys.ucl.ac.be
- ✓ If you have any questions, comments or suggestions on this tool, feel free to contact me.
- ✓ If you want the download the latest version of the application, you can get it from www.isys.ucl.ac.be/staff/Adrien/SketchiXML

14. Acknowledgements

- ✓ We gratefully acknowledge the support of the Request research project under the umbrella of the WIST (Wallonie Information Société Technologies) program under convention n°031/5592 RW REQUEST).
- ✓ We warmly thank J.A. Jorge and his team for allowing us to use JavaSketchIt and the CALI library.

References

A

[Abra99]

Abrams M, Phanouriou C., Batongbacal A. L. , Williams S., Shuster J., *UIML: An Appliance-Independent XML User Interface Language*, in Computer Networks: The International Journal of Computer and Telecommunications Networking, v.31 n.11-16, p.1695-1708, May 17, 1999.

[Alva04a]

Alvarado C. and Davis R., *SketchREAD: A Multi-domain Sketch Recognition Engine*, in Proceedings of User Interface Software and Technology UIST'04, pp. 23-32, Santa Fe, USA 2004.

[Alva04b]

Alvarado C., *Sketch Recognition User Interfaces: Guidelines for Design and Development*, AAAI Fall Symposium on Intelligent Pen-based Interfaces, 2004.

[Arid98]

Aridor Y. and Lange D. B., *Agent Design Patterns: Elements of Agent Application Design*, in Proceedings of the 2nd Int. Conf. on Autonomous Agents Agents'98, pp. 108-115 Minneapolis, USA, 1998.

[Arth86]

Arthur J. D., Nance R. E., and Henry S. M., *A Procedural Approach to Evaluating Software Development Methodologies: the Foundation*, Technical Report. UMI Order Number: TR-86-24., Virginia Polytechnic Institute & State University, 1986.

[Axur]

Axure RP: <http://www.axure.com/products.aspx>

B

[Bail03]

Bailey B.P. and Konstan J.A., *Are Informal Tools Better? Comparing DEMAIS, Pencil and Paper, and Authorware for Early Multimedia Design*, in Proceedings of the ACM Conference on Human Factors in Computing Systems CHI'03, pp. 313-320, Fort Lauderdale, USA, April 2003.

References

- [Baum96] Baümer D., Bischofberger W.R., Lichter H., and Züllighoven H., *User Interface Prototyping - Concepts, Tools, and Experience*, in Proceeding of 18th International Conference on Software Engineering ICSE'96, pp. 532-541, Berlin, Germany, 1996.
- [Beck87] Beck K., and Cunningham W., *Using Pattern Languages for Object-Oriented Programs*, in Workshop on the Specification and Design for Object-Oriented Programming OOPSLA'87, 1987.
- [Brat88] M. Bratman, *Intentions, Plans and Practical Reasoning*, Harvard Univ. Press, 1987.
- [Bede94] Bederson B.B. and Hollan J.D., *Pad++: A Zooming Graphical Interface for Exploring Alternative Interface Physics*, in Proceedings of the ACM Symposium on User Interface Software and Technology UIST '94, pp. 17-26, Marina del Rey, USA, 1994.
- [Berk00] Berkun, S. The Art of User Interface Prototyping, November 2000, accessible at <http://www.scottberkun.com/essays/essay12.htm>
- [Blan04] Blankenhorn K., Jeckle M., *A UML Profile for GUI Layout*, in Proceeding of Net.ObjectDays'04, pp. 110-121, Erfurt, Germany, 2004.
- [Bush96] Buschmann F., Meunier R., Rohnert H., Sommerlad P. and Stal M., *Pattern-Oriented Software Architecture - A System of Patterns*, John Wiley & Sons, Inc., New York, USA, 1996.
- [Boar84] Boar, B.H., *Application prototyping: a requirements definition strategy for the 80s*, John Wiley & Sons, Inc., New York, USA, 1984.
- [Bodk91] Bodker S. and Gronbaek K., *Cooperative prototyping: users and designers in mutual activity*, in Computer-Supported Cooperative Work and Groupware, pp. 331-358, S. Greenberg, Ed. Academic Press Computers And People Series, Academic Press Ltd., London, UK, 1991.
- [Boeh88] Boehm W., *A Spiral Model of Software Development and Enhancement*, Computer, IEEE communication 21 n.5, pp.61-72, 1988.
- [Bord81] Borda J. C., *Memoire sur les elections au scrutiny*, Histoire de l'Academie Royale des Sciences, 1781.
- [Bres65] Bresenham, J.E., *Algorithm for Computer Control of a Digital Plotter*, IBM Systems Journal, Vol. 4, No. 1, pp. 25-30, 1965.

References

- [Brow97]
Brown J., *Exploring Human-Computer Interaction and Software Engineering. Methodologies for the Creation of Interactive Software*, in *SIGCHI Bulletin* 29(1), pp. 32–35, 1997.
- C
- [Caet02]
Caetano A., Goulart N., Fonseca M. and Jorge J., *JavaSketchIt: Issues in Sketching the Look of User Interfaces*, in *Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding*, pp. 9-14, Palo Alto, USA, 2002.
- [Car]
GUI Design Studio: <http://www.carettasoftware.com/gds/index.html>
- [Calv97]
Calvary G., Coutaz J., Nigay L., *From single-user architectural design to PAC*: a generic software architecture model for CSCW*, in *Proceedings of the SIGCHI conference on Human factors in computing systems*, p.242-249, Atlanta, USA, 1997.
- [Chat99]
Chatty S. and Dewan P. (Eds), *Engineering for Human Computer Interaction*, Kluwer Academics, 1999.
- [Chat03]
Chatty S., Sire S. and Lemort A., *Vers des outils pour les équipes de conception d'interfaces post-WIMP*, In *Proceedings of the 16th Conference on Association Francophone D'interaction Homme-Machine (Namur, Belgium, August 30 - September 03, 2004)*. IHM 2004. ACM Press, New York, NY, 45-52.
- [Clem99]
Clements P. C., *Constructing Superior Software*, Sams, 1999.
- [Cock01]
Cockburn, A., *Writing Effective Use Cases*, Addison-Wesley, 2001.
- [Cons99]
Contantine L. and Lockwood L., *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional, 1999.
- [Cout87]
Coutaz J., *PAC, an Object Oriented Model for Dialog Design*, in *Human-Computer Interaction H.-J. Bullinger and B. Shackel (Eds.) Elsevier Science Publishers B.V.*, pages 431 - 436, 1987.
- [Coye03]
Coyette A., *The SkwyRL-Agent Architectural Framework : Developing An E-Business Application*, M.Sc. thesis, UCL, Louvain-la-Neuve, 2003.
- [Coye05]
Coyette, A., Vanderdonckt, J.: A Sketching Tool for Designing Anyuser, Anyplatform, Anywhere User Interfaces. In: *Proc. of 10th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2005 (Rome, September 12-16, 2005)*. Lecture Notes in Computer Science, Vol. 3585. Springer-Verlag, Berlin (2005) 550–564.

References

- [Coye07]
Coyette, A., Kieffer, S. Vanderdonckt, J., Multi-fidelity Prototyping of User Interfaces, in Proc. of 11th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2007 (Rio de Janeiro, September 10-14, 2007), Lecture Notes in Computer Science, Springer-Verlag, 2007, to appear.
- D**
- [Dear03]
Dearden A, Siddiqi J. & Naghsh A., *Using Cognitive Dimensions to Compare Prototyping Techniques*, In Proceedings of the 15th Annual Workshop of the Psychology of Programming Interest Group. 8th - 10th April, 2003.
- [Deug99]
Deugo D., Oppacher F., Kuester J. and Otte I. V., *Patterns as a Means for Intelligent Software Engineering*, In Proceedings of the International Conference on Artificial Intelligence ICAI'99, pp. 605-611, Las Vegas, USA, 1999.
- [Dsou99]
D'Souza, D.F., and Wills, A.C., *Objects, Components and Frameworks with UML: The Catalysis Approach*, Addison-Wesley, Reading, 1999.
- [Do05]
Do T.T., *A Social Patterns Framework for Designing Multiagent Architectures*, Ph.D. thesis, Université catholique de Louvain, IAG, Louvain-la-Neuve, June 2005.
- E**
- [Easy]
EasyPrototype: <http://www.extremeplanner.com/easyprototype/>
- F**
- [Faul04a]
Faulkner S., Kolp M., Coyette A. and Do T. T., *Agent-Oriented Design of E-Commerce Style Architectures*, In Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS 2004), Porto, Portugal, pp 372-379, April 2004.
- [Faul04b]
Faulkner S., *An Architectural Framework for Describing BDI Multi-Agent Information Systems*, Ph.D. Thesis, Université Catholique de Louvain, Institut d'Administration et de Gestion (IAG), Louvain-la-Neuve, Belgium, May 2004.
- [Fern01]
Fernandez E. B. and Pan R., *A Pattern Language for Security Models*, In Proceedings of PLoP, Monticello, USA, 2001.
- [Fipa]
Foundation for Intelligent Physical Agents - <http://www.fipa.org/>
- [Fons02]
Fonseca M.J., Pimentel C. and Jorge J.A., *CALI: An Online Scribble Recognizer for Calligraphic Interfaces*, in Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding, pp. 51-58, Palo Alto, USA, 2002,

References

[Free74]

Freeman, H., *Computer Processing of Line-Drawing Images*, ACM Computing Surveys, Vol. 6, No. 1, pp. 57-97, 1974.

G

[GHJ95]

Gamma E., Helm R., Johnson R. and Vlissides J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1995.

[Gros96]

Gross M. D. and Do E., *Ambiguous Intentions: A Paperlike Interface for Creative Design*, in Proceedings of ACM Symposium on User Interface Software and Technology, UIST '96, pp. 183-192, Seattle, USA, 1996.

H

[Hayd99]

Hayden S., Carrick C. and Yang Q., *Architectural Design Patterns for Multiagent Coordination*, In Proceedings of the 3rd International Conference on Agent Systems Agents'99, Seattle, USA, 1999.

[High01]

Highsmith J., Cockburn A., *Agile Software Development: The Business of Innovation*, Computer, v.34 n.9, p.120-122, September 2001

[Hint62]

Hintikka J., *Knowledge and belief*, Cornell University Press, 1962.

[Hong01]

Hong J.I., Li F.C., Lin J., and Landay J.A., *End-User Perceptions of Formal and Informal Representations of Web Sites*, in Extended Abstracts of Proceedings of ACM Conference on Human Factors in Computing Systems CHI 2001, Seattle, USA, 2001.

J

[Jack]

JACK Intelligent Agents. <http://www.agent-software.com/>.

[Jadex]

Jadex BDI Agent Systems <http://www.informatik.uni-hamburg.de/projects/jadex/>.

[Jenn01]

Jennings N. R. and Wooldridge M., *Agent-Oriented Software Engineering*, In Handbook of Agent Technology AAAI/ MIT Press, 2001.

[Juch04]

Juchmes R. and Leclercq P., *A Multi-Agent System for Architectural Sketches Interpretation*, Eurographics Workshop on Sketch-Based Interfaces, Grenoble, France, 2004

K

[Kend99]

References

Kendall K.E. & Kendall J., *System Analysis and Design*, 3rd edition, Prentice Hall, 1999.

[Kolp01]

Kolp M., Giorgini P. and Mylopoulos J., *A Goal-Based Organizational Perspective on Multi-agent Architectures*, Revised Papers from the 8th International Workshop on Intelligent Agents VIII, pp.128-140, August 01-03, 2001.

[Kono82]

Konolige K., *A first order formalization of knowledge and action for multi-agent planning system*, in Machine Intelligence, 10, pp. 41-72, 1982.

L

[Land95]

Landay J.A., Myers B. A., *Interactive Sketching for the Early Stages of User Interface Design*, In Proceedings of CHI '95: Human Factors in Computing Systems, pp. 43-50, Denver, USA, May 1995.

[Land96]

Landay J.A., *Interactive Sketching for the Early Stages of User Interface Design*,. Ph.D. thesis, report #CMU-CS-96-201, Computer Science Department, Carnegie Mellon University, Pittsburgh, 1996.

[Land01]

Landay J., Myers B.A., *Sketching Interfaces: Toward More Human Interface Design*, IEEE Computer 34, vol. 3, pp. 56-64, 2001.

[Leve66]

Levenshtein V.I., *Binary codes capable of correcting deletions, insertions, and reversals*, Doklady Akademii Nauk SSSR, Vol. 163, No. 4, 1965, pp. 845-848 [in Russian]. English translation in Soviet Physics Doklady, Vol. 10, No. 8, pp. 707-710, 1966.

[Lew95]

Lewis J.R., *Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for use*, in International Journal of Human-Computer Interaction 7(1), pp. 57-78, 1995.

[Lim06]

Lim Y., Pangam A., Periyasami S., and Aneja S., *Comparative analysis of high- and low-fidelity prototypes for more valid usability evaluations of mobile devices*, In Proceedings of the 4th Nordic Conference on Human-Computer interaction: Changing Roles (Oslo, Norway, October 14 - 18, 2006). A. Mørch, K. Morgan, T. Bratteteig, G. Ghosh, and D. Svanaes, Eds. NordiCHI '06, vol. 189. ACM Press, New York, NY, 291-300.

[Lin2000]

Lin J., Newman M. W., Hong J.I., and Landay J.A., *DENIM: Finding a tighter fit between tools and practice for web site design*, in Proceedings of the SIGCHI conference on Human factors in computing systems CHI '00, pp. 510-517, The Hague, The Netherlands.

[Limb04a]

Limbouq Q. and Vanderdonckt J., *Transformational Development of User Interfaces with Graph Transformations*, in Proceedings of 5th International Conference on Computer-Aided Design of User Interfaces CADUI'2004, pp. 105-118, Madeira, Portugal, 2004.

References

- [Limb04b] Limbourg Q., Vanderdonckt J., Michotte B., Bouillon L., Florins M., and Trevisan D., *USIXML: A User Interface Description Language for Context-Sensitive User Interfaces*, In Proceedings of the first international Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages, Gallipoli, Italy, 2004.
- [Limb04c] Limbourg Q., Vanderdonckt J., Michotte B., Bouillon L., and Lopez-Jaquero V., *USIXML: a Language Supporting Multi-Path Development of User Interfaces*, In Proceedings of 9th IFIP Working Conference on Engineering for Human-Computer Interaction EHCI-DSVIS'04, Hamburg, 2004.
- [Limb05] Limbourg Q., *Multi-Path Development of User Interfaces*, Ph.D. thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, 2005.
- [Luo95] Luo P., *A Human-Computer Collaboration Paradigm for Bridging Design Conceptualization and Implementation*, in Proceedings of the First International Eurographics Workshop, pp. 129–147, Bocca di Magra, Italy, 1994.
- M**
- [Mccu06] McCurdy M., Connors C., Pyrzak G., Kanefsky B., and Vera A., *Breaking the fidelity barrier: an examination of our current characterization of prototypes and an example of a mixed-fidelity success*, In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (Montréal, Québec, Canada, April 22 - 27, 2006). R. Grinter, T. Rodden, P. Aoki, E. Cutrell, R. Jeffries, and G. Olson, Eds. CHI '06. ACM Press, New York, NY, 1233-1242.
- [Meye96] Meyer J., *EtchaPad – Disposable Sketch Based Interfaces*, in Proceedings of Human Factors in Computing Systems (Conference Companion), ACM CHI '96, pp. 195-198, Vancouver, Canada, 1996.
- [Meye05] Meyer J.: *Creating Informal Looking Interfaces*, 2005, accessible at http://www.cybergrain.com/tech/pubs/lines_technote.html.
- [Micr07] Microsoft Visio: <http://office.microsoft.com/en-us/visio/HA101656401033.aspx>
- [Mock] MockUpScreens: <http://www.mockupscreens.com/>
- [Mora06] Mora R., Juchmes R., Rivard H. & Leclercq P., 2006, From an architectural sketch to feasible structural systems, Second International Conference on Design Computing & Cognition, TU/e Eindhoven, Pays-Bas.
- [Myer00] Myers B., Hudson S. , and Pausch R., *Past, present, future of user interface tools*, ACM Transactions on Computer-Human Interaction (TOCHI), 7(1), pp. 3-28, 2000.

N

References

- [Nana95]
Nanard J. and Nanard M., *Hypertext design environments and the hypertext design process*, Communications of the ACM, v.38 n.8, p.49-56, Aug. 1995.
- [Nawr05]
Nawrocki J.R., Olek L., UC Workbench: A Tool for Writing Use-Cases, 6th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2005, Sheffield, UK, June 18-23, 2005, Lecture Notes in Computer Science 3556, Springer, pp 230-234.
- [Newm03]
Newman M. N., Lin J., Hong J. I., and Landay J. A., *DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice*, in Human-Computer Interaction, 18(3), pp. 259-324, 2003.
- O**
- [OASIS]
Organization for the Advancement of Structured Information Standards (<http://www.oasis-open.org/>)
- P**
- [Pala97]
Palanque P. and Paterno F., *Formal Methods in Human-Computer Interaction*, Springer-Verlag New York, Inc., Secaucus, NJ, 1997
- [Pate00]
Paternò, F., *Model-Based Design and Evaluation of Interactive Applications*, Springer-Verlag, Berlin, 2000.
- [Paec03]
Paech B., Kohler K., *Usability Engineering Integrated with Requirements Engineering*, in Proceedings of ICSE'03 (International Conference for Software Engineering), pp 36-40, Portland, USA, 2003.
- [Plim02]
Plimmer B.E., Apperley M., *Computer-aided sketching to capture preliminary design*, In Proceedings of the Third Australasian conference on User interfaces - Volume 7, Australian Computer Society, Inc., pp. 9-12, Melbourne Australia, 2002.
- [Plim03]
Plimmer B.E., Apperley M., *Software for Students to Sketch Interface Designs*, In Proceedings of IFIP Conference on Human-Computer Interaction Interact'03, pp. 73-80, Zurich, Switzerland, 2003.
- [Plim04]
Plimmer B., and Apperley M., *Interacting with Sketched Interface Designs: An Evaluation Study*, in Proceedings of ACM Conference on Human Factors in Computing Systems CHI'04, pp. 1337-1340, Vienna, April Austria 2004.
- [Plim07]
InkKit : <http://www.cs.auckland.ac.nz/~beryl/inkkit.html>
- [Puer97]

References

Puerta A.R., *A Model-Based Interface Development Environment*, in IEEE Software 14(4), pp. 41-47, 1997.

[Puer99]

Puerta, A. and Eisenstein, J., *Towards a General Computational Framework for Model-Based Interface Development Systems Model-Based Interfaces*, in Proceedings of 3rd International ACM Conference on Intelligent User Interfaces IUI'99 Redondo Beach, USA, pp. 171-178, 1999.

[Puer05]

Puerta A.R., Micheletti M, Mak A., *The UI pilot: a model-based tool to guide early interface design*, in Proceedings of the 2005 International Conference on Intelligent User Interfaces IUI 2005, San Diego, USA, pp. 215-222, 2005.

R

[Red]

RedWhale: <http://www.redwhale.com>

[Rett94]

Rettig M., *Prototyping for tiny fingers*, Communications of the ACM, April, Vol.37, No.4 (1994).

[Rudd96]

Rudd, J., Stern, K. and Isensee, S. (1996) *Low vs. high fidelity prototyping debate. Interactions*, V.3 n1, p76-85, ACM Press.

S

[Shaw96]

Shaw M. and Garlan D., *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996.

[Schi06]

Schimke S., Vielhauer C. and Dittmann J., *Using Adapted Levenshtein Distance for On-Line Signature Authentication*, in Proceedings of 17th International Conference on the Pattern Recognition(ICPR'04), pp. 931-934, 2004.

[Sinh03]

Sinha, A. K. and Landay, J. A. 2003. *Capturing user tests in a multimodal, multidevice informal prototyping tool*. In Proceedings of the 5th international Conference on Multimodal interfaces (Vancouver, British Columbia, Canada, November 05 - 07, 2003). ICMI '03. ACM Press, New York, NY, 117-124.

[Sieg88]

Siegel S. and Castellan Jr. N. J., *Nonparametric Statistics for The Behavioral Sciences*, McGraw-Hill, Inc., second edition, 1988.

[Snyd02]

Snyder, C., *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces*. Morgan Kaufmann, April, 2002.

[Sumn97]

Sumner T., Bonnardel N. and Harstad Kallak B., *The Cognitive Ergonomics of Knowledge-Based Design Support Systems*, in Proceedings of ACM Conference on Human Aspects in Computing Systems CHI'97, pp. 83-90, Atlanta, USA, 1997.

References

- [Sun98]
<http://java.sun.com/developer/technicalArticles/ALT/Reflection/>
- [StpBA]
stpBA Storyboarding : <http://www.stpsoft.co.uk/story/index.html>
- [Szek96a]
Szekely P., Sukaviriya P., Castells J., Muthukumarasamy J., and Salcher E., *Declarative interface models for user interface construction tools: The MASTERMIND approach*, in Engineering for Human-Computer Interaction, Chapman & Hall, pp. 120-150, London, 1996.
- T**
- [Tohi06]
Tohidi M., Buxton W. Baecker, R. and Sellen, A., *User sketches: a quick, inexpensive, and effective way to elicit more reflective user feedback*, In Proceedings of the 4th Nordic Conference on Human-Computer interaction: Changing Roles (Oslo, Norway, October 14 - 18, 2006). A. Mørch, K. Morgan, T. Bratteteig, G. Ghosh, and D. Svanaes, Eds. NordiCHI '06, vol. 189. ACM Press, New York, NY, 105-114.
- U**
- [Ucwo]
<http://www.ucworkbench.org/>
- [Unge96]
Unger C. and Bass L., *Engineering for HCI*, Kluwer Academics Publishers, 1996.
- V**
- [Vand97]
Vanderdonckt J., *Conception Assistée de la Présentation D'une Interface Homme-Machine Ergonomique Pour Une Application de Gestion Hautement Interactive*. PhD thesis, Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique, Namur, 1997.
- [Vand02]
van Duyne D.K., Landay J.A. and Hong J.I., *The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience*, Addison-Wesley 2002.
- [Vand04]
Vanden Berghe Y., *Etude et implémentation d'un générateur d'interfaces vectorielles à partir d'un langage de description d'interfaces utilisateur*, M.Sc. thesis, Université catholique de Louvain, Louvain-la-Neuve, Belgium, 2004.
- [Vand06]
Vanderdonckt J., Coyette A., Vers un prototypage des interfaces graphiques incluant vraiment l'utilisateur final, in Proceedings of 10ième Colloque International sur l'Ergonomie et l'Informatique Avancée ERGO-IA'2006 (Biarritz, 11-13 October 2006), E. Brangier, Ch. Kolski, J.-R. Ruault (eds.), Ecole Supérieure des Technologies Industrielles Avancées (ESTIA/ILS), Bidart, pp. 31-42, 2006.

References

- [Vand06b]
Vanden Bossche P., *Développement d'un outil de critique d'interface intelligent: UsabilityAdviser*, M.Sc. thesis, UCL, Louvain-la-Neuve, 2006.
- [Virz96]
Virzi R.A., Sokolov, J.L., Karis, D.: *Usability problem identification using both Low- and High-Fidelity Prototypes*. In: Proceedings of ACM Conference on Human Aspects in Computing Systems CHI'96, Vancouver, Canada, pp. 236-243, 1996.
- W**
- [War3]
Warcraft 3: <http://www.blizzard.com/war3/>
- [Walk02]
Miriam Walker, Leila Takayama, James A. Landay, "High-fidelity or low-fidelity, paper or computer medium?" In the Proceedings of the Human Factors and Ergonomics Society 46th Annual Meeting, Baltimore, pp. 661-665, October 2002.
- [Wegn97]
Wegner P., *Why Interaction is more Powerful than Algorithms*, in Communications of the ACM, 40(5), pp. 80-91, 1997.
- [Wool96]
Wooldridge M. and Jennings N.R., *Special Issue on Intelligent Agents and Multi-Agent Systems*, in Applied Artificial Intelligence Journal, 9(4), pp.74–86, 1996.
- X**
- [Xiao02]
Xiaogang X., Liu W., Xiangyu J. and Zhengxing S., *Sketch-based User Interface for Creative Tasks*, in Proceedings of 5th Asia Pacific Conference on Computer Human Interaction(APCHI2002) , pp. 560-570, Beijing, China, 2002.
- [XUL]
[http:// www.mozilla.org/projects/xul/](http://www.mozilla.org/projects/xul/)
- Y**
- [Yu95]
Yu E., *Modeling Strategic Relationships for Process Reengineering*, Ph.D. thesis, Department of Computer Science, University of Toronto, Toronto, 1995.
- [Yu01]
Yu E., *Agent-Oriented Modelling: Software Versus the World*, In Proceedings of the 2nd International Workshop on Agent-Oriented Software Engineering, Lecture Notes in Computer Science 2222, pp 206-225, Springer Verlag, 2001