

A Sketching Tool for Designing Anyuser, Anyplatform, Anywhere User Interfaces

Adrien Coyette and Jean Vanderdonckt

Université catholique de Louvain, School of Management (IAG),
Place des Doyens, 1 – B-1348 Louvain-la-Neuve, Belgium
{coyette, vanderdonckt}@isys.ucl.ac.be

Abstract. Sketching activities are widely adopted during early design phases of user interface development to convey informal specifications of the interface presentation and dialog. Designers or even end users can sketch some or all of the future interface they want. With the ever increasing availability of different computing platforms, a need arises to continuously support sketching across these platforms with their various programming languages, interface development environments and operating systems. To address needs along these dimensions, which pose new challenges to user interface sketching tools, SketchiXML is a multi-platform multi-agent interactive application that enable designers and end users to sketch user interfaces with different levels of details and support for different contexts of use. The results of the sketching are then analyzed to produce interface specifications independently of any context, including user and platform. These specifications are exploited to progressively produce one or several interfaces, for one or many users, platforms, and environments.

1 Introduction

Designing the right User Interface (UI) the first time is very unlikely to occur. Instead, UI design is recognized as a process that is [15] intrinsically *open* (new considerations may appear at any time), *iterative* (several cycles are needed to reach an acceptable result), and *incomplete* (not all required considerations are available at design time). Consequently, means to support early UI design has been extensively researched [16] to identify appropriate techniques such as paper sketching, prototypes, mock-ups, diagrams, etc. Most designers consider hand sketches on paper as one of the most effective ways to represent the first drafts of a future UI [1,8,10]. Indeed, this kind of unconstrained approach presents many advantages: 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 (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 the contract and validation. Van Duyne *et al.* [16] reported that 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). The idea of developing a computer-based tool for sketching UIs naturally emerged from these observations [6,12]. 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 [15] such as checking and revision. Some research was carried out in order to propose a hybrid approach, combining the best of the hand-sketching and computer assisted interface design, but this marriage highlights five shortcomings:

- Some tools only support sketching activities, without producing any output: when 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.
- Sketching tools that recognize the drawing do produce some output, but not in a reusable format: the design output is not necessarily in a format that is directly reusable as development input, thus preventing reusability.
- Sketching tools are bound to a particular programming language, a particular UI type, a particular computing platform or operating system: when an output is produced, it is usually bound to one particular environment, therefore preventing developers from re-using sketches in new contexts, such as for various platforms.
- Sketching tools do not take into account the sketcher's preferences: as they impose the same sketching scheme, the same gestures for all types of sketchers, a learning curve may prevent these users from learning the tool and efficiently using it.
- Sketching tools do not allow a lot of flexibility in the sketch recognition: the user cannot choose when recognition will occur, degrading openness [15] and when this occurs, it is difficult to return to a previous state.

To unleash the power of informal UI design based on sketches, we need to address the above shortcomings observed for existing UI sketching tools. The expectation is thus that UI sketching will be lead to its full potential. SketchiXML is a new informal prototyping tool solving *all* these shortcomings, letting designers sketch user interfaces as easily as on paper. In addition, SketchiXML provides the designer with on-demand design critique and assistance during early design. Instead of producing code specific to a particular case or environment, SketchiXML generates UI specifications written in UsiXML (User Interface eXtensible Markup Language – www.usixml.org), a platform-independent User Interface Description Language (UIDL) that will be in turn exploited to produce code for one or several UIs, and for one or many contexts of use simultaneously.

In this paper Section 2 demonstrates that state-of-the-art UI sketching tools all suffer from some of the above shortcomings. Section 3 reports on an experimental study conducted to identify the sketchers' preferences, such as the most preferred and appropriate UI representations. These results underpin the development of SketchiXML in Section 4, where these widgets are recognized on demand. The multi-agent architecture of SketchiXML is outlined to support various scenarios in different contexts of use with examples. Section 5 discusses some future work and Section 6 demonstrates that the seven shortcomings above are effectively solved in SketchiXML.

2 Related Work

UI prototypes usually fall into three categories depending on their degree of fidelity, that is the precision to which they reproduce the reality of the desired UI.

The *high-fidelity* (Hi-Fi) prototyping tools support building a UI that looks complete, and might be usable. Moreover, this kind of software is equipped with a wide range of editing functions for all UI widgets: erase, undo, move, specify physical attributes, etc... This software lets designers build a complete GUI, from which is produced an accurate image (e.g., Adobe Photoshop, PowerPoint) or code in a determined programming language (e.g., Visual Basic, DreamWeaver). Even if the final result is not executable, it can still be considered as a high fidelity tool given that the result provided looks complete.

The *medium-fidelity* (Me-Fi) approach builds UI mock-ups giving importance to content, but keeping secondary all information regarding typography, color scheme or others minor details. A typical example is Microsoft Visio, where only the type, the size and the contents of UI widgets can be specified graphically.

Low-fidelity (Lo-Fi) drafting tools are used to capture the general information needed to obtain a global comprehension of what is desired, keeping all the unnecessary details out of the process. The most standard approaches for Lo-Fi prototyping are the “paper and pencil technique”, the “whiteboard/blackboard and post-its approach” [16]. Such approaches provide access to all the components, and prevent the designer from being distracted from the primary task of design. Research shows that designers who work out conceptual ideas on paper tend to iterate more and explore the design space more broadly, whereas designers using computer-based tools tend to take only one idea and work it out in detail [6,12,15]. Many designers have reported that the quality of the discussion when people are presented with a Hi-Fi prototype was different than when they are presented with a Lo-Fi mock up. When using Lo-Fi prototyping, the users tend to focus on the interaction or on the overall site structure rather than on the color scheme or others details irrelevant at this level [16].

Consequently, Lo-Fi prototyping offers a clear set of advantages compared to the Hi-Fi perspective, but at the same time suffers from a lack of assistance. For instance, if several screens have a lot in common, it could be profitable to use copy and paste instead of rewriting the whole screen each time. A combination of these approaches appears to make sense, as long as the Lo-Fi advantages are maintained. This consideration results two families of software tools which support UI sketching and representing the scenarios between them, one with and one without code generation.

DENIM [6,10] helps web site designers during early design by sketching information at different refinement levels, such as site map, story board and individual page, and unifies the levels through zooming views. DEMAIS [1] is similar in principle, but aimed at prototyping interactive multimedia applications. It is made up of an interactive multimedia storyboard tool that uses a designer's ink strokes and textual annotations as an input design vocabulary. Both DENIM and DEMAIS use pen input as a natural way to sketch on screen, but do not produce any final code or other output.

In contrast, SILK [8], JavaSketchIt [2] and Freeform [11,12] are major applications for pen-input based interface design supporting code generation. SILK uses pen input to draw GUIs and produce code for OpenLook operating system. JavaSketchIt proceeds in a slightly different way than Freeform, as it displays the shapes recognized in

real time, and generates Java UI code. JavaSketchIt uses the CALI library [6] for the shape recognition, and widgets are formed on basis of a combination of vectorial shapes. The recognition rate of the CALI library is very high and thus makes JavaSketchIt easy to use, even for a novice user. Freeform only displays the shapes recognized once the design of the whole interface is completed, and produces Visual Basic 6 code. The technique used to identify the widgets is the same than JavaSketchIt, but with a slightly lower recognition rate. Freeform also supports scenario management thanks to a basic storyboard view similar to that provided in DENIM.

Table 1. Comparison of software for low-, medium-, and high-fidelity UI prototyping tools

Fidelity	Appearance	Advantages	Shortcomings
Low	<ul style="list-style-type: none"> - Sketchy - Little visual detail 	<ul style="list-style-type: none"> - Low development cost - Short production time - Easy communication - Basic drawing skills needed 	<ul style="list-style-type: none"> - Is facilitator-driven - Limited for usability tests - Limited support of navigational aspects - Low attractiveness for end users - No code generation
Medium	<ul style="list-style-type: none"> - Simple - medium level of detail, close to appearance of final UI 	<ul style="list-style-type: none"> - 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 	<ul style="list-style-type: none"> - Is facilitator-driven - Limited for usability tests - Medium support of navigational aspects - No code generation
High	<ul style="list-style-type: none"> - Definitive, refined - Look and Feel of final UI 	<ul style="list-style-type: none"> - Fully interactive - Serves for usability testing - Supports user-centered design - Serves for prototype validation and contract - Attractive for end users - Code generation 	<ul style="list-style-type: none"> - High development cost - High production time - Advanced drawing and specification skills needed - Very inflexible with respect to changing requirements

Table 1 summarizes major advantages and shortcomings of existing UI prototyping tools depending on their level of fidelity. In addition to the shortcomings in the last column, the shortcomings outlined in the introduction should also be considered to elicit the requirements of SketchiXML. SketchiXML’s main goal is to combine in a flexible way the advantages of the tools just presented into a single application, but also to add new features for this kind of application. Thus SketchiXML should avoid the five shortcoming above by: (R1) producing UI specifications and generate from them UI in several programming languages to avoid binding with a particular environment and to foster reusability; (R2) supporting UI sketching with recognition and translation of this sketching into UI specifications in order not to loose the design effort; (R3) supporting sketching for any context of use (e.g., any user, any platform, any environment) instead of only one platform, one context; (R4) being based on UI widget representations that are significant for the designer and/or the end-user; and

(R5) performing sketch recognition at different moments, instead of at an imposed moment. R4 is addressed in Section 3, the others, in Section 4.

Others vital facilities to be provided by SketchiXML are handling input from different sources (R6), such as direct sketching on a tablet or a paper scan, and also receiving real time advice on two types of issues (R7), if desired: the first occurs in a post-sketching phase, and provides a set of usability advice based on the UI drawn. For the second type of advice, the system operates in real time, looking for possible patterns, or similarities with previously drawn UIs. The objective of such an analysis is to supplement the sketching for the designer when a pattern is detected. Since the goal of SketchiXML is to entice designers to be creative and to express evaluative judgments, we infer the rules enunciated in [15] to the global architecture, and let the designer parameterizes the behavior of the whole system through a set of parameters (Section 4).

3 Building the Widgets Catalogue

This section presents the method used to define the widget catalogue. The first subsection introduces the method itself. Subsection 2 provides a short analysis of the results.

3.1 Method

To address requirement R3, SketchiXML recognizes different representations, different sketches for the same UI widget. Indeed, the advantage of such a tool lies in the fact that it imitates the informality of classical low-fidelity tools, and is thus required to be easy and natural to use. For this purpose, we have conducted an experimental study aimed at collecting information on how users intuitively sketch widgets. Two groups of 30 subjects were randomly selected from a list: the first group had relevant experience in the computer science domain and interface design, while the second were end users with no specific knowledge of UI design or computer science. The second group was considered because SketchiXML's goal is to involve the end user as much as possible in the early prototyping process to bridge the gap between what they say and what the designer understands. Thus, the representations may vary between designers and end users. Fig. 1 depicts the various domains of expertise of each group.

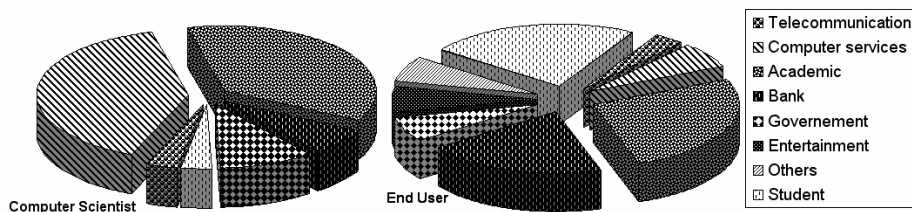








Fig. 1. Distribution of the subjects according to their domain of expertise

A two phase 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 catalogue was identified comprising the following 32 widgets: text, text field, text area, push button, search field, login, logout, reset form, validate, radio button, check box, combo box, image, multimedia area, layer, group box, table, separator, frame, hyperlink, anchor, list box, tabbed dialog box, menu, color picker, file picker, date picker, hour picker, toggle button, slider, progress bar, spinner. Each widget was documented with its English and French name, a screen shot and a small textual definition (see first three columns of Table 2). For each widget, subjects were asked if they had ever seen this widget before and to provide a sketching representation. Then, from the widget representations provided during the first phase, we tried in a second phase to extract the most common representations, in order to build a second questionnaire. In this questionnaire, 5 representations were associated with each widget, and participants were asked to rank the different representations (last column of Table 2) according to their representativeness and preference as a five point Likert scale. Fig. 2 depicts the propositions for a list box that will be examined as a representative example in the next subsection.

Table 2. Definition of the widgets catalogue (excerpts)

Widget	Graphical presentation	Textual definition	Potential sketchings
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.	
Date Picker		This widget allows the user to pick a date on an agenda.	

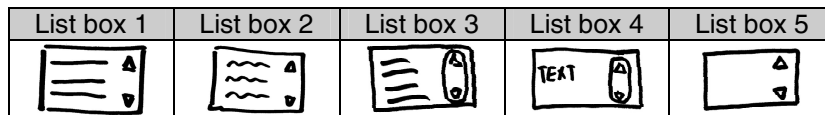


Fig. 2. Sketching propositions for the list box widget

3.2 Results and Discussion

Due to space restriction, we mainly focus on the list box widget. Based on the result distribution showed in Fig 3, we establish the best representation with the following method. Firstly we assess whether any dependence exists between the participants. If this first step's results established a significant dependence, then we proceed to the second phase and we compute the aggregate preference of both groups and the global preference. For each widget, the Kendall coefficient of concordance W 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.

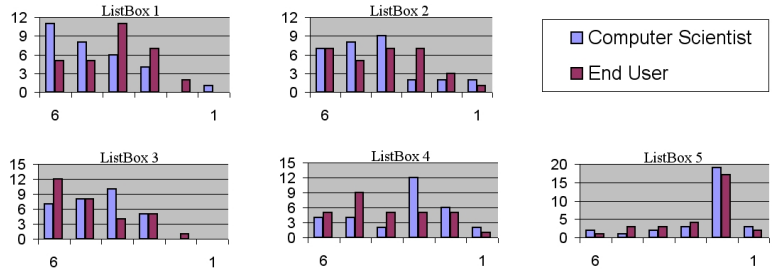


Fig. 3. Result frequency of the survey regarding the list box

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

Fig. 4. Computation of W where k is the number of judges, N the number of objects being ranked, R_i the average of the ranks assigned to the i^{th} object, 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

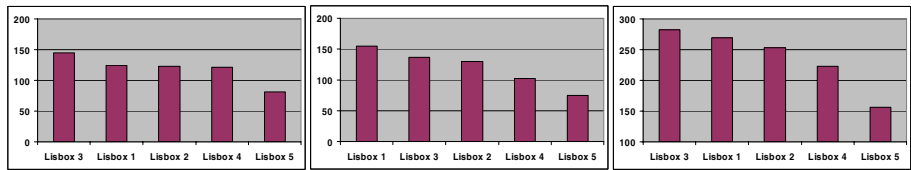


Fig. 5. Borda Count results for end users, computer scientists and both categories aggregated

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 [14]. 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 basis of this analysis we observed that both groups have almost the same preferences among the representations (Fig. 5). Most of the time, the set of well considered representations is the same, even if small changes in the sequence occur. Out of this set, we chose preferred representations on the basis of intrinsic complexity, which is defined on basis of a set of criteria such as the number of strokes, the need of new vectorial shapes, high probability of confusion with other widget... For instance, list box 4 obtained a good score compared to the other representations, but its intrinsic complexity is very high, since it requires hand writing recognition, that is not supported for the moment. List box 4 and 5 were thus discarded from the final selection. Often, representations selected for the list box are composed from the three first representations in Figure 2.

4 SketchiXML Development

After meeting requirement R3 in the previous section, we have to address the remaining requirements, i.e. the application has to carry out shape recognition (R2), provide spatial shape interpretation (R2), provide usability advice (R7), handle several kinds of input (R6), generate UsiXML specifications (R1), and operate in a flexible way (R5). To address these requirements, a BDI (*Belief-Desire-Intention*) agent-oriented architecture [4] was considered appropriate: such architecture allows building robust and flexible applications by distributing the responsibilities among autonomous and cooperating agents. Each agent is in charge of a specific part of the process, and cooperates with the others in order to provide the service required according to the designer's preferences. This kind of approach has the advantage of being more flexible, modular and robust than traditional architecture including object-oriented ones [4].

4.1 SketchiXML Architecture

The application was built using the SKwyRL-framework [7], a framework aimed at defining, formalizing and applying socially based catalogues of styles and patterns to construct agent and multi-agent architectures. The joint-venture organizational style pattern [7] was applied to design the agent-architecture of SketchiXML [3]. It was chosen on basis of non-functional requirements R_i , as among all organizational styles defined in the SKwyRL framework, the joint venture clearly matches the requirements defined in Section 2 as the most open and distributed organizational style.

The architecture (Fig. 6) is structured using i^* [17], 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.

When a user wishes to create a new SketchiXML project, she contacts the *Broker* agent, which serves as an intermediary between the external actor and the organizational system. The *Broker* queries the user for all the relevant information needed for the process, such as the target platform, the input type, the intervention strategy of the *Adviser* agent,... According to the criteria entered, the coordinator chooses the most suitable handling and coordinates all the agents participating in the process in order to meet the objectives determined by the user. For clearness, the following section only considers a situation where the user has selected real time recognition, and pen-input device as input. So, the *Data Editor* agent then displays a white board allowing the user to draw his hand-sketch interface. All the strokes are collected and then transmitted to the *Shape Recognizer* agent for recognition. The recognition engine of this agent is based on the CALI library [5], a recognition engine able to identify shapes of different sizes, rotated at arbitrary angles, drawn with dashed, continuous strokes or overlapping lines.

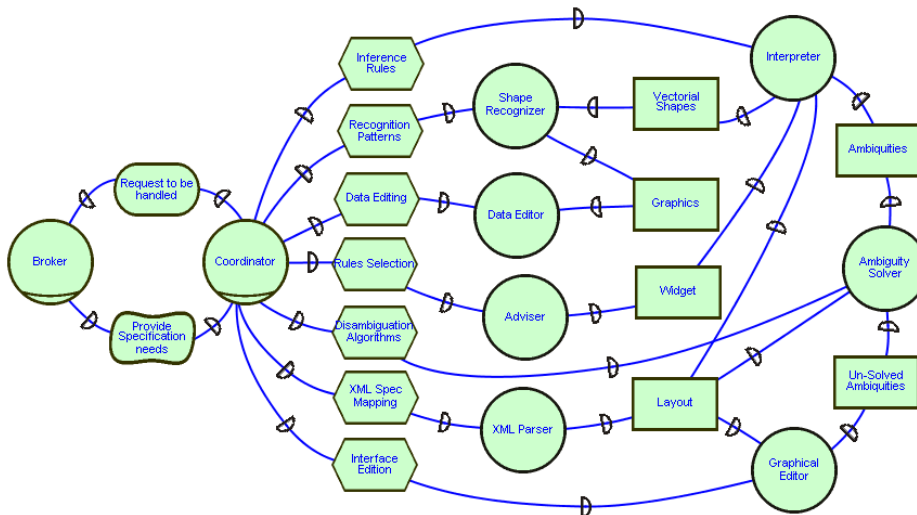


Fig. 6. i* representation of SketchiXML architecture as a Joint-Venture

Subsequently, the *Shape Recognizer* agent provides all the vectorial shapes identified with relevant information such as location, dimension or degree of certainty associated to the *Interpreter* agent. Based on these shape sets, the *Interpreter* agent attempts to create a component layout. The technique used for the creation of this layout takes advantage of the knowledge capacity of agents. The agent stores all the shapes identified as his belief, and each time a new shape is received all the potential candidates for association are extracted. Using its set of patterns the agent then evaluates if shape pairs form a widget or a sub-widget. The conditions to be tested are based on a set of fuzzy spatial relations allowing to deal with imprecise spatial combinations of geometric shapes and to fluctuate with user preferences. Based on the widgets identified by the *Interpreter*, the *Adviser* agent assists the designer with the conception of the UIs in two different ways.

Firstly, by providing real-time assistance to the designer by attempting to detect UI patterns in the current sketch in order to complete the sketch automatically. Secondly in a post operational mode, the usability adviser provides usability advice on the interface sketched. If the *Interpreter* fails to identify all the components or to apply all the usability rules, then the *Ambiguity Solver* agent is invoked. This agent evaluates how to solve the problem according to the initial parameters entered by the user.

The agent can either attempt to solve the ambiguity itself by using its set of disambiguation algorithms, or to delegate it to a third agent, the *Graphical Editor* agent. The *Graphical Editor* displays all the widget recognized at this point, as classical element-based software, and highlights all the components with a low degree of certainty for confirmation. Once one of these last three agents evoked has sufficient certainty about the overall widget layout, the UI is sent to the *XML Parser* agent for UsiXML generation.

4.2 Low-Fidelity Prototyping with SketchiXML

The first step in SketchiXML consists of specifying parameters that will drive the low-fidelity prototyping process (Fig. 7): the project name, the input type (i.e. on-line sketching or off-line drawing that is scanned and processed in one step-Fig. 8), the computing platform for which the UI is prototyped (a predefined platform can be selected such as mobile phone, PDA, TabletPC, kiosk, ScreenPhone, laptop, desktop, wall screen, or a custom one can be defined in terms of platform model [9]), the output folder, the time when the recognition process is initiated (ranging from on-demand manual to fully automatic each time a new widget can be detected- this flexibility is vital according to experiments and [15]), the intervention mode of the usability advisor (manual, mixed-initiative, automatic), and the output quality stating the response time vs. quality of results of the recognition and usability advisor processes. In Fig. 7, the UsiXML parsing is set on fully manual mode, and the output quality is set on medium quality. The quality level affects the way the agents consider a widget

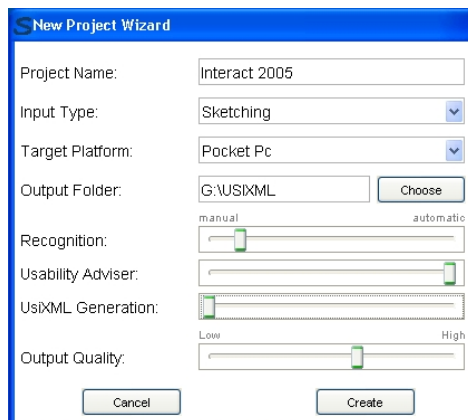


Fig. 7. Creating a new SketchiXML prototype

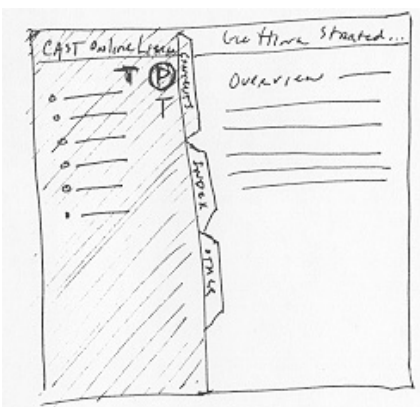


Fig. 8. Scanned UI sketching

layout to be acceptable, or the constraints used for the pattern matching between vectorial shapes. The sketching phase in that situation is thus very similar to the sketching process of an application such as Freeform [11]. Of course, the designer is always free to change these parameters while the process is running.

Fig. 9 illustrates the SketchiXML workspace configured for designing a UI for a standard personal computer. On the left part we can observe that shape recognition is disabled as none of the sketches is interpreted, and the widget layout generated by the *Interpreter* agent remains empty. The right part represents the same UI with shape recognition and interpretation. Fig. 10 depicts SketchiXML parameterized for a PocketPC platform and its results imported in GrafiXML, a UsiXML-compliant graphical UI editor that can generate code for HTML, XHTML, and Java (<http://www.usixml.org/index.php?view=page&idpage=10>).

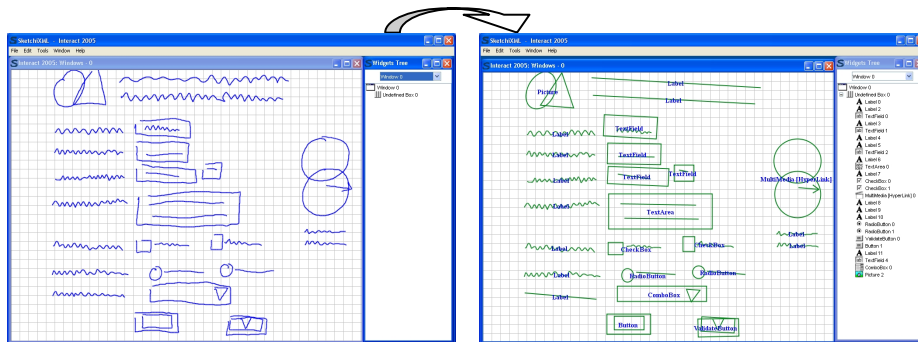


Fig. 9. SketchiXML workspace

When shape recognition is activated, each time a new widget is identified the color of the shapes turns to green, and the widget tree generated by the *Interpreter* is updated. Changing the context has a deep impact on the way the system operates. As an example, when a user builds a user interface for one platform or another, adaptations need to be based on the design knowledge that will be used for evaluation, by selecting and prioritizing rule sets [15], and on the set of available widgets. As the size of the drawing area is changing, the set of constraints used for the interpretation needs to be tailored too, indeed if the average size of the strokes drawn is much smaller than on a standard display, the imprecision associated with each stroke follows the same trend. We can thus strengthen the constraints to avoid any confusion.

Once the design phase is complete, SketchiXML parses the informal design to produce UsiXML specifications. Fig. 11 gives an overview of the UsiXML specifications generated from UI drawn in Fig. 10. Each widget is represented with standard values for each attribute, as SketchiXML is only aimed at capturing the UI core properties. In addition, the UsiXML specifications integrate all the information related to the context of use as specified in the wizard depicted on Fig. 7: information for the user model, the platform model, and the environment model [9]. As UsiXML allows defining a set of transformation rules for switching from one of the UsiXML models to another, or to adapt a model for another context, such information is thus required.

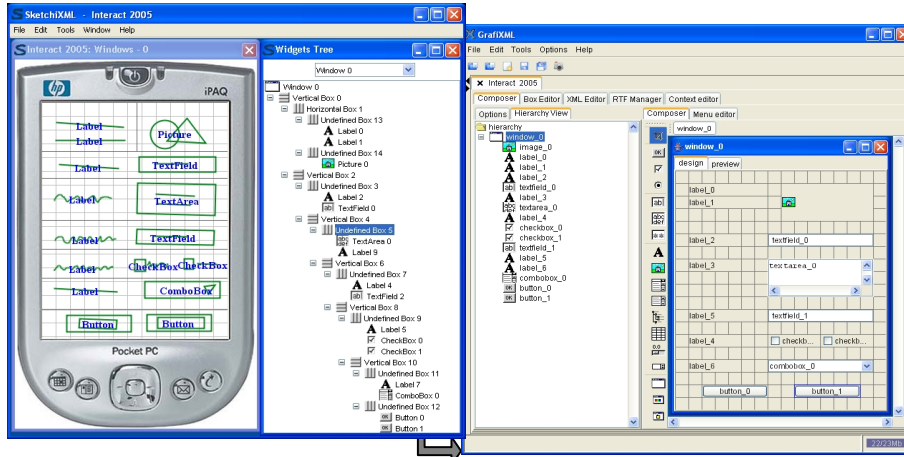


Fig. 10. SketchiXML workspace configured for a PDA and its import in GrafixXML

```
<?xml version="1.0" encoding="UTF-8"?>
<uiModel id="Interact_2005" name="Interact 2005"
creationDate="2005-01-06T14:51:31.656+01:00" schemaVersion="1.6.1"
xmlns="http://www.usixml.org">
<version modifDate="2005-01-06T14:51:31.656+01:00" xmlns="">1</version>
<authorName xmlns="">SketchiXML</authorName>
<uiModel id="Interact_2005_6-cui" name="Interact 2005-cui">
<window id="window_0" name="window_0" isVisible="true"
isEnabled="true" fgColor="#000000" bgColor="#e9e9d8" borderWidth="0" width="400"
height="350"
isAlwaysOnTop="false" windowLeftMargin="0" windowTopMargin="0" isResizable="true">
<box id="box_0" name="box_0" isVisible="false"
isEnabled="true" width="400" height="350" type="horizontal" isFlow="false"
isFill="false"
isScrollable="false" isSplittable="false" isDetachable="false"
isResizableVertical="false"
isResizableHorizontal="false" relativeMinWidth="0" relativeMinHeight="0"
isBalanced="false"
relativeWidth="0" relativeHeight="0">...
<textComponent id="label_0" name="label_0"
isVisible="true" isEnabled="true" fgColor="#000000" bgColor="#e9e9d8"
visitedLinkColor="#000000"
activeLinkColor="#000000" isBold="true" isItalic="false" isUnderline="false"
isStrikethrough="false" isSubscript="false" isSuperscript="false"
isPreformatted="false"
textColor="#000000" textSize="12" textFont="Dialog" textMargin="0"
textVerticalAlign="middle"
textHorizontalAlign="left" scrollStyle="scroll" scrollDirection="left"/>...
<imageComponent id="image_0" name="image_0" isVisible="true" isEnabled="true"
defaultHyperLinkTarget="">...
<button id="button_1" name="button_1" isVisible="true" isEnabled="true"
fgColor="#000000" bgColor="#e9e9d8"/>...
</box> </window>
</uiModel>
<contextModel id="interact_2005-contextModel_0" name="interact_2005-contextModel_pda">
<context id="interact_2005-context_0" name="interact_2005-context_pda">
<userStereotype id="interact_2005-user_US_0" language="en_US"
stereotypeName="interact_2005-user_US"
taskExperience="1" systemExperience="1" deviceExperience="1" taskMotivation="1"/>
<platform id="windows_mobile_2003" name="windows_mobile_2003">
<softwarePlatform OSName="Windows" OSVersion="2003" OSVendor="Microsoft Corp."/>
<hardwarePlatform screenSize="240x320" />
</platform></context></contextModel>
<resourceModel id="Interact_2005_6" name="Interact 2005"/>
</uiModel>
```

Fig. 11. Excerpt of the UsiXML specifications generated by SketchiXML

Fig. 10 illustrates the SketchiXML output imported in GrafiXML, a high fidelity UI graphical editor. On basis of the informal design provided during the early design, a programmer can re-use the output without any loss of time to provide a revised version of the UI with all the characteristics that can and should not be defined during the early design phase. This contrasts with a traditional approach, where a programmer had to implement user interfaces on basis of a set of blackboard photographs or sheets of paper, and thus start the implementation process from the beginning.

As the Usability Advisor intervention time has been specified as “automatic” (Fig. 7), each time a usability deviation is detected with respect to usability guidelines, a tool tip message is produced in context, attached to the widget on concern. For this purpose, a set of form-based usability guidelines have been encoded in GDL (Guideline Definition Language), a XML-compliant description of guidelines that can be directly related to UsiXML widgets.

5 Future Work

Although SketchiXML already provides a wide set of features, many evolutions could be imagined. Out of many ideas, three major ones retain our attention:

1. One drawback of SketchiXML is the lack of a scenario editor allowing to represent transition between screen. Capturing such information could be very profitable, and is quite natural to represent for a novice designer. Moreover such information can be directly stored in the UsiXML model and be reused just as easily as the code generated for each UI.
2. A second high potential evolution consists in developing an evolutionary recognition engine. SketchiXML uses the CALI library [5] and a set of spatial constraints between the vector shapes recognized to build the widget. Even if the recognition rate is very high, the insertion of new widget representation is restricted to a combination of the set of the vector shapes supported. To this aim, research in a biometric domain such as handwriting recognition [13] could provide valuable answers, taking full advantage of the multi-agent architecture.
3. During the sketching process, the possibility to instantly switch to a runnable version of the current UI is useful. Indeed, all informal design tools providing code generation allow easy switching from design to run mode, while SketchiXML requires to invoke a third application. Right now, SketchiXML only supports import in GrafiXML. So, we would like to support existing external interpreters that produce Flash, Java, XHTML and Tcl-Tk interpretations (see www.usixml.org for a list of such interpreters)

6 Conclusion

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, the application is based on a BDI multi-agent architecture where each requirement is assumed by an autonomous and

collaborative agent part of an organizational system. Based on the criteria provided by the designer at the beginning of the process, the experts (agents) adapt the way they act and interact with the designer and the other agents in order to meet the global objectives. We have shown that SketchiXML meets requirements R1-R5 that were identified as important shortcomings of existing tools. Through this research, we have also conducted a survey on 60 people from different activity sectors with different backgrounds, in order to identify how these people would intuitively represent the widgets to be handled by SketchiXML. From these results we have associated a set of sketching representations to each widget. Moreover, this set of representation is not hard coded and can be reconfigured by the user through an external configuration file. SketchiXML will extend a set of tools initiating the design process from the early design phase to the final concrete user interface, with tools supporting every stage. The complete widgets catalogue, screen shots, demonstration of SketchiXML and implementation are available at www.usixml.org. SketchiXML is developed in Java, on top SKwyRL-framework [7] and JACK Agent platform, with recognition based on CALI library [5].

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, F.M.G. Pereira and A. Caetano for allowing us to use JavaSketchIt and the CALI library in our research, Mickaël Nicolay for conducting the user survey and providing the results, and Gilbert Cockton for helping us in the preparation of this manuscript.

References

1. Bailey, B.P., Konstan, J.A.: Are Informal Tools Better? Comparing DEMAIS, Pencil and Paper, and Authorware for Early Multimedia Design. In: Proc. of the ACM Conf. on Human Factors in Computing Systems CHI'2003. ACM Press, NY (2003) 313–320
2. Caetano, A., Goulart, N., Fonseca, M., Jorge, J.: JavaSketchIt: Issues in Sketching the Look of User Interfaces. In: Proc. of the 2002 AAAI Spring Symposium - Sketch Understanding (Palo Alto, March 2002). AAAI Press (2002) 9–14
3. Coyette, A., Faulkner S., Kolp, M., Vanderdonckt, J., Limbourg, Q.: SketchiXML: Towards a Multi-Agent Design Tool for Sketching User Interfaces Based on USIXML. In: Proc. of TAMODIA'2004 (Prague, November 2004). ACM Press, New York (2004) 75–82
4. Faulkner, S.: An Architectural Framework for Describing BDI Multi-Agent Information Systems. Ph.D. Thesis, UCL-IAG, Louvain-la-Neuve (May 2004)
5. Fonseca, M.J., Jorge, J.A.: Using Fuzzy Logic to Recognize Geometric Shapes Interactively. In: Proc. of the 9th Int. Conf. on Fuzzy Systems FUZZ-IEEE'00 (San Antonio, 2000). IEEE Computer Society Press, Los Alamitos (2000) 191–196
6. Hong, J.I., Li, F.C., Lin, J., Landay, J.A.: End-User Perceptions of Formal and Informal Representations of Web Sites. In: Extended Abstracts of CHI'2001, 385–386

7. Kolp, M., Giorgini, P., Mylopoulos, J.: An Organizational Perspective on Multi-agent Architectures. In: Proc. of the 8th Int. Workshop on Agent Theories, Architectures, and Languages ATAL'01 (Seattle, 2001).
8. Landay, J., Myers, B.A.: Sketching Interfaces: Toward More Human Interface Design. *IEEE Computer* 34, 3 (March 2001) 56–64
9. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Lopez-Jaquero, V. USIXML: a Language Supporting Multi-Path Development of User Interfaces. In: Proc. of 9th IFIP Working Conf. on Engineering for Human-Computer Interaction EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). Kluwer Academics, Dordrecht (2004)
10. Newman, M.W., Lin, J., Hong, J.I., Landay, J.A.: DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice. *Human-Comp. Interaction* 18 (2003) 259–324
11. Plimmer, B.E., Apperley, M. Software for Students to Sketch Interface Designs. In: Proc. of IFIP Conf. on Human-Computer Interaction INTERACT'2003. IOS Press (2003) 73–80
12. Plimmer, B.E., Apperley, M.: Interacting with Sketched Interface Designs: An Evaluation Study. In: Proc. of CHI'04. ACM Press, New York (2004) 1337–1340
13. Schimke S., Vielhauer C., Dittmann J.: Using Adapted Levenshtein Distance for On-Line Signature Authentication. In: Proc. of ICPR'2004. Springer-Verlag (2004) 931–934
14. Sidney Siegel and Jr. N. John Castellan. *Nonparametric Statistics for The Behavioral Sciences*. McGraw-Hill, Inc., second edition, 1988.
15. Sumner, T., Bonnardel, N., Kallag-Harstad, B., The Cognitive Ergonomics of Knowledge-based Design Support Systems. In: Proc. of CHI'97. ACM Press, New York (1997) 83–90
16. van Duyne, D.K., J.A. Landay, and J.I. Hong, *The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience*. Addison-Wesley (2002).
17. Yu, E.: *Modeling Strategic Relationships for Process Reengineering*. Ph.D. thesis. Department of Computer Science, University of Toronto, Toronto (1995).