Prototyping Digital, Physical, and Mixed User Interfaces by Sketching

Adrien Coyette, Jean Vanderdonckt

Université catholique de Louvain, Louvain School of Management, Place des Doyens, 1
B-1348 Louvain-la-Neuve (Belgium)
{adrien.coyette, jean.vanderdonckt}@uclouvain.be

ABSTRACT

Sketching digital and physical user interfaces present many benefits such as naturalness, intuitiveness, ability to elicit user requirements, and ability to discover usability problems. These advantages are confirmed in MIXEDSKETCH, a software for prototyping any type of user interface by sketching: a digital interface such as a graphical user interface, a physical interface such as a tangible user interface, and, more uniquely, mixed user interfaces that combine elements from both digital and physical worlds, simultaneously or at different design stages. As the development process proceeds from early design to detailed development, MIXEDSKETCH ensures a smooth transition from a low-fidelity representation to a high-level representation of the UI being sketched through mid-fidelity. In the last stage, a precise presentation and a dialog can be sketched that automatically generate a description of the future interface for one or multiple toolkits. In addition, MIXEDSKETCH enables the designer to transform a digital, a physical, or a mixed user interface in a counterpart in another world, e.g. moving from digital to physical to mixed.

Author Keywords
Level of fidelity, shape recognition, sketching, user interface design, user interface prototyping.

General Terms
Design, Human Factors, Languages.

ACM Classification Keywords
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *User interfaces*. H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *Graphical user interfaces*. I.3.6 [**Computer Graphics**]: Methodology and Techniques – *Interaction techniques.*

INTRODUCTION

Prototyping a digital user interface (UI), such as a Graphical User Interface (GUI), is typically conducted in a GUI builder by dragging interaction objects or widgets (such as radio buttons, push buttons, and sliders) from a palette and dropping them onto a working surface area. In order to fine tune the *presentation* (i.e., the UI static part), the designer specifies physical properties of these widgets such as location, dimensions, colors, and layout. When it comes to defining the *dialog* (i.e., the UI dynamic part), the drawing visual paradigm stops: the designer needs to open every widget of concern and program its behavior in the programming, markup or scripting language supported by the Integrated Development Environment (IDE).

Prototyping a physical User Interface (UI), such as a Tangible User Interface (TUI) [4], is even more challenging: the objects are by definition physical, thus difficult to prototype unless a digital counterpart exists. Both their presentation and their dialog are complex to specify, existing IDEs offer little or no help, everything is often coded by hand, leaving little or no space for prototyping outside coding.

We hereby define a *Mixed User Interface* (MUI) as a UI that combines digital and physical objects to be designed and developed in a coordinated way, by ensuring consistency across both worlds and integrating them [10]. MUI prototyping is even more complex since there is a need to bridge the gap between softwired and hardwired worlds [25] and make a correspondence between them.

Sketching is universally recognized for its natural [11], unconstrained [27], and informal [9] virtues in multiple areas of human activity, such as layout design [12], multimedia design [1], and UI design [3,5,11,16,20]. As long as the sketching is not submitted to any recognition engine, the end user does not perceive any shortcoming apart from little or no reusability of the sketches for future steps in the design process [18]. When the sketches need to be recognized, e.g. for beautification [3] or interpretation [22], the end user may feel again constrained as she knows that every gesture should be performed correctly in order to be properly recognized, thus reducing the virtue of naturalness.

UI design by sketching has already demonstrated several advantages: UI sketching is preferred over traditional interface builders, especially by end users [11,16], it can be performed at different levels of fidelity without loosing advantages [17,26], the amount of usability problems discovered through a sketched design is not inferior to those corresponding to a genuine UI [26], the expressive power of a sketched UI remains the same [27], a sketched UI provides quantitative and qualitative results that are comparable to traditional UI prototypes except that the cost is reduced [18], UI sketching encourages exploratory design and fosters communication between stakeholders more than any other prototypes [22], flexibility is superior to UI builders [27], authoring tools [1], and paper prototypes [27]. Therefore, sketching is an interaction technique that could be effectively and efficiently used for prototyping UIs, but has never been considered extensively for designing GUIs, TUIs, and mixed UIs at the same time.

There are a number of problems that traditional sketching methods pose which are challenging for novice users and inefficient for expert users. The first problem is related to the meaningfulness of representations: what is the best object representation? Should multiple representations of the same object be offered? How should it be sketched? Should it be sketched in one stroke or several strokes? If a representation is not meaningful enough for an end user, the representation will be forgotten or badly drawn. The second problem is that the result of the chosen representation is often far from what is expected by the novice user and difficult to reproduce [16]. The third problem with traditional recognition engines is that the representations should be different enough [12] and sketched precisely enough to be efficiently recognized [3,9,12].

To address these problems, we developed MIXEDSKETCH, a multi-platform sketching tool that provides original functionalities with respect to state-of-the-art software (as described in the following section) and that supports designing GUIs, TUIs, and MUIs. These three UI types will be addressed respectively in the next sections, as well as the link between. A summary of the main contributions of this paper and a description of the planned follow-up work conclude the paper.

RELATED WORK

During the UI development life cycle, the design step is often characterized as a process that is intrinsically open (new designs may appear at any time that require further exploration), iterative (several cycles are performed to reach a solution), and incomplete (not all information is available at design time) [11]. The area of UI design by sketching has been extensively researched to identify appropriate techniques such as paper sketching, prototypes, mock-ups, diagrams [1,3,5,9,11,13,16,20]. Several software for UI design by sketching emerged from this research: DENIM [13], DEMAIS [1], FreeForms [16], InkKit [16], JavaSketchIt [5], Satin [9], Silk [11] to name the most representative ones.

Since the needs of rapid UI prototyping vary depending on the project and allocated resources, it makes sense to rely on the level of fidelity. The level of fidelity expresses the similarity between the final UI and the prototyped UI. The UI prototype fidelity is said to be *high* if the prototype representation is the closest possible to the final UI, or almost in the same representation (based on [16]). This means that the prototype should be of high-fidelity in terms of presentation (what layout, what are the UI elements used), of global navigation and dialog (how to navigate between information spaces), of local navigation (how to navigate within an information space). More precisely, McCurdy *et al.* [14] identified five independent dimensions along which the level of fidelity could be more rigorously defined: the level of visual refinement, the breadth of functionality, the depth of functionality, the richness of interactivity, and the richness of the data model. In the remainder of this paper, the four first dimensions will be considered, the last one requiring a connection to a data model containing data.

Similarly to the above definition, the level of fidelity is said to be *low* if the prototype representation only partially evokes the final UI without representing it in full details. Between *high-fidelity* (Hi-Fi) and *low-fidelity* (Lo-Fi) [17], we can introduce *medium-fidelity* (Me-Fi). We usually observe that a UI prototype only involves one representation type, i.e. one fidelity level at a time. But due to the variety of stakeholders' inputs, several levels of fidelities could be combined together, thus leading to the concept of *mixed-fidelity*, where several different fidelities are mixed in the same UI design [13]. Beyond mixed-fidelity, we introduce *multi-fidelity* that is reached when a prototype simultaneously involves elements belonging to different levels of fidelity, but only one level of fidelity is acted upon at a time, thus assuming that a transition is always possible for an element from one level of fidelity to another.

Another system to support UI design on tangible surfaces is shown in The Designer's Outpost [10]. The system tracks multiple post-it notes on a surface and is used to support the designing of web sites. Several post-its are added, labeled and recognized by computer-vision. Electronic sketching is hereby used to add hierarchy to the post-its on the surface. Digital pen input is used to describe functional dependencies between the different post-its.

Similarly to The Designer's Outpost, the Pin&Play environment [3,23,25], the VoodooIO toolkit [25,26] support the arrangement of multiple physical TUI elements on a rigid or flexible surface area and connect them togeth so as to form a working interface, e.g. for music [28], for sound design [27] based on a TUI. However, instead of using passive post-its, the aforementioned systems system use active elements like buttons, sliders, or dials.

Successful examples of prototyping environments for various UI types are demonstrated their feasibility and applicability: DEMAIS for multimedia applications [1], iStuff for ubiquitous computing [2], JavaSketchIt for Java applications [4], d.Tools for TUIs [8], The Designer's Outpost for web sites [10], PALETTE for presentations [14], PaperButtons for physical UIs [15], PaperPoint [19], and VoodooFlash for TUIs [21].

In the next sections, we will demonstrate that MIXEDSKETCH departs from the state-of-the-art, in particular the aforementioned works, in that it:

- Supports simultaneously the rapid prototyping of digital UIs (such as GUIs), physical UIs (such as TUIs), and mixed UIs (MUIs) that combine elements belonging to both worlds, digital and physical.
- Enables designers to sketch these UIs in multiple levels of fidelity, to transform each element from one level of

fidelity to another, and to apply fusion of elements.

- Enables designers to smoothly switch from one world to another in order to prototype a behaviorally-equivalent UI in another world (for instance, a GUI as a MUI).
- Exports a representation of such UIs that can be imported in a IDE, an appropriate toolkit, or rendering engine.

DIGITAL INTERFACE PROTOTYPING

This section describes how MIXEDSKETCH supports the prototyping life cycle of a GUI belonging to the digital world. The original features of the tool are progressively motivated, described, and discussed. This sketching tool today consists of about 122,000 lines of Java 1.5 code and can be freely downloaded from www.anonymized.org, both the executable software and its full source code. This tool enables designers to sketch a UI as easily as on paper, while keeping advantages of computer-based design as identified and confirmed in [1,16,26,27]. At any time, the designer may ask the tool to recognize the UI being sketched and generate a working UI from the sketches. At any time, it provides the facilities detailed in the following subsections.

World modeling. MIXEDSKETCH models the design space as a simplified object-oriented world, i.e. a set of *objects* linked by typed *relationships*. Each object can be described by its location (X,Y), its dimensions (length and height), and additional properties that can be specified through property sheets (e.g., design annotations, color). Each relationship is characterized by its type (e.g., spatial constraint, decomposition, navigation link), by its edges (i.e., how many source and target edges, what types of objects are accepted for each edge), and by additional properties (e.g., one-way or two-ways transitions). Therefore, both digital and physical design worlds will be represented in the same way with connections between.

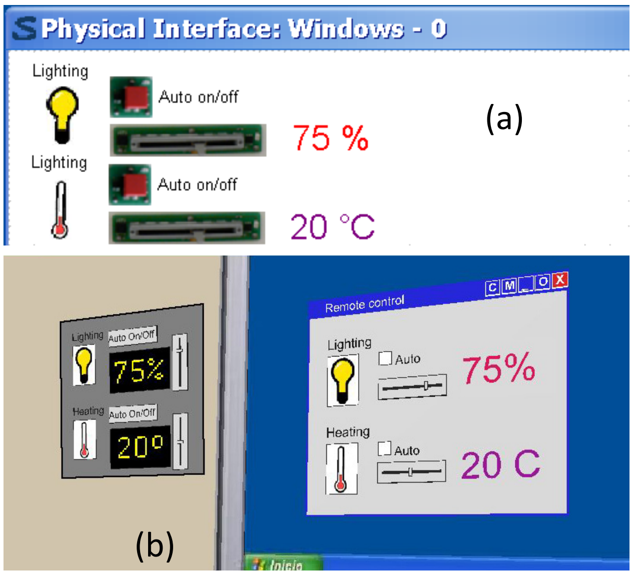


Figure 1. Three UIs for lighting and heating control: a TUI (a), a physical UI (b, left), a software GUI (b, right).

Fig. 1 depicts three different UIs for a simple home control of lighting and heating: a TUI sketched in terms of the Voodoo toolkit [4] in MIXEDSKETCH (a), a physical UI made of physical devices (b, left), and a GUI made of corresponding widgets in Windows environment (b, right) rendered in a 3D scene. Each time a UI is designed, its underlying model is automatically generated for the end user in an appropriate User Interface Description Language (UIDL).

Object recognition. An object recognition engine recognizes and interprets 32 different types of widgets (ranging from check boxes and spin button to search buttons, progress bar, calendar, and video input), 8 basic predefined shapes (i.e., triangle, rectangle, cross, line, wavy line, arrow, ellipse, and circle), and 6 basic commands (i.e., undo, redo, copy, paste, cut, new window). This amount of recognized objects is superior to what can be found in other software equipped with a recognition engine in the same domain [3,11]. Each object is rigorously defined in terms of constituent shapes (any of the 8 aforementioned basic shapes) and constraints between them.

Each constraint should belong to the set of the 31 constraints supported today: areParallel, cross, hasInside, hasInsideInLowerRightCorner, hasInsideInTheCenter, hasInsideInTop, hasInsideInUpperRightCorner, hasInsideOnTheLeft, hasInsideOnTheRight, hasPositiveSlope, intersect, isCrossedBy, isHorizontal, isInside, isInsideInBottom, isInsideInLowerRightCorner, isInsideInTheCenter, isInsideInTop, isInsideInUpperRightCorner, isInsideOnTheLeft, isInsideOnTheRight, isOnTheLeftOf, isOnTheRightOf, isOnUpperLeftCorner, isSmall, isSquare, isThin, isUnder, isVertical. Any object representation is expressed in a XML format stored in a graphical grammar [3] that is parsed and interpreted at run-time [11]. In this way, any custom object could be easily added by adding a new representation in the grammar. Each UI element can be sketched and recognized or not depending on its shape and the wish for the user to see it recognized or not. The object recognition is only on-demand. Those shapes which are not recognized are simply added and maintained throughout the process. Fig. 2 shows a design session where some UI objects have been sketched in Lo-Fi mode. In this mode, objects which have been correctly recognized are beautified and the name is added.

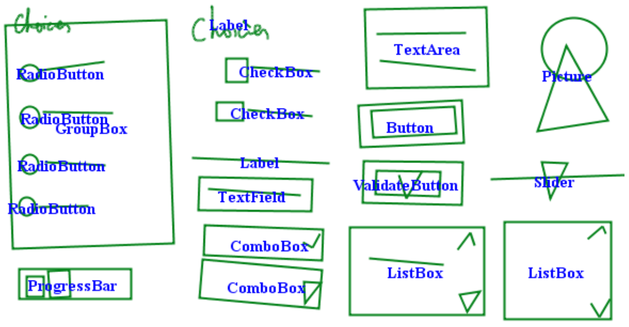


Figure 2. A typical GUI design session with sketched objects.

Multiple object representations. Existing software incorporating an object recognition engine typically support only one single representation per object, most frequently through a mono-directional single-stroke gesture [3]. Our tool accommodates several representations for a single object, without affecting significantly the system response time. Fig. 3 reproduces an excerpt of the radio button representation. In addition, thanks to this logical definition, each representation could be sketched in a multi-stroke manner that is independent of the direction. In this way, left-handed or right-handed persons are equally supported.

```
<widget type="RadioButton">
<representation id="0">
  <constraint id="0" shape1="Line_1"
  shape2="Circle_0" condition="isOnTheRightOf"
  />
  <constraint id="1" shape1="Line_1" shape2="-"
  condition="isHorizontal" />
  <shape id="Circle_0" type="Circle" />
  <shape id="Line_1" type="Line" />
  </representation>
  …
</widget>
```

Figure 3. A representation for the radio button.

Multi-fidelity representation. Thanks to the object recognition process, the designer can input any UI object in any level of fidelity and see the result in any other level as the interpretation is immediate. In the same way, any custom object could be drawn in Lo-Fi and a predefined widget could be added in Me-Fi or Hi-Fi. Therefore, four fidelity levels are supported as recommended by [14]: none (only the drawing is displayed), Lo-Fi (the drawing is displayed with recognized portions), Me-Fi (the drawing is beautified where portions are recognized, including for basic shapes), and Hi-Fi (a genuine UI is produced with widgets for those recognized portions). Fig. 4 exemplifies multi-fidelity representations for a subset of the 32 widgets supported. To our knowledge, no existing software today supports so many widgets in different levels of fidelity as we have here.

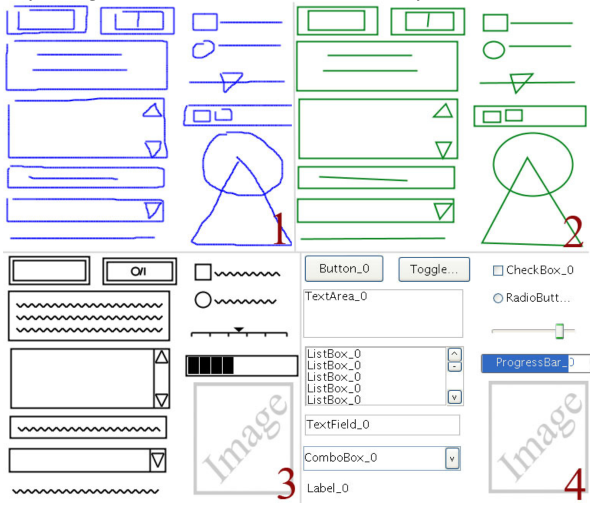


Figure 4. A set of widgets with the representations corresponding to the four levels of fidelities (none, low, medium, high).

Fidelity transition. A slider allows the designer to easily switch between the four levels of fidelity (Fig. 5). Fig. 6 shows a GUI prototype after the designer moved from No-Fi (a) to Lo-Fi (b), to Me-Fi (c), and Hi-Fi (d). Note that the representation of Fig. 6c is platform agnostic: it does not produce any representation that would suggest any particular window manager or UI builder. In Fig. 6d, the representation is made up of genuine widgets belonging to the widget set of the currently being used platform (here, Java). In order to identify the parts recognized by the engine, labels of recognized widgets can be turn on (as in Fig. 6b and d) or off (as in Fig. 6a and c). Different widget sets and look & feel could be used alternatively that mimic a Hi-Fi representation in other window managers and operating systems. If a UI element has not been recognized, it is simply kept as it is. For instance, if a histogram would have been sketched, it would not be altered so as to respect the naturalness of the design process as recommended in [13].

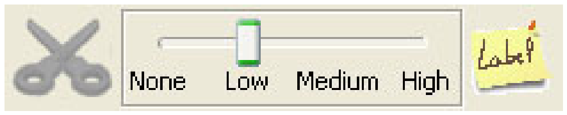
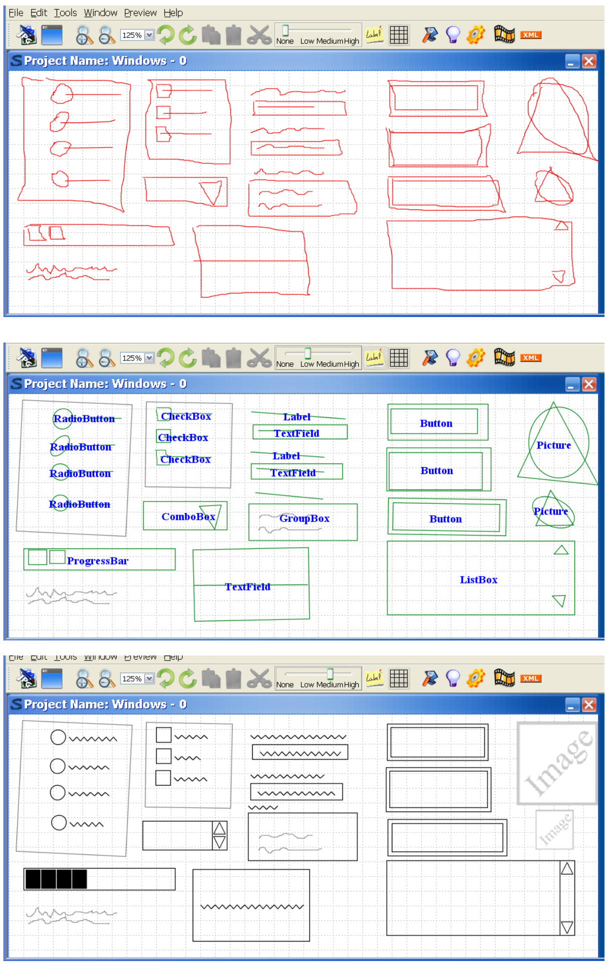


Figure 5. Slide to switch between levels of fidelity.



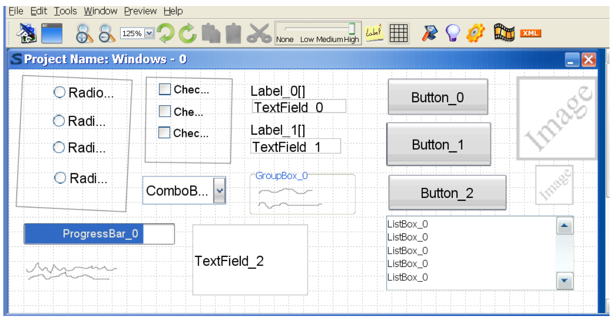


Figure 6. Transition between the four levels of fidelity.

Gesture recognition. Sketching tool users sometimes complained that they are forced to learn a graphical representation [12] for every widget, shape or command. In order to support this user flexibility, each such object could of course give rise to a new representation in the graphical grammar. Some user studies revealed the need for the user to interactively define personalized representations [12]. For this purpose, a gesture recognition system has been implemented based on hand gesture decomposition in order to customize the representation of all widgets, shapes, and commands according to each user's preferences (Fig. 7). One or several occurrences of a new gesture could be graphical defined that will then serve as a redundant input technique for every widget, shape, or command.

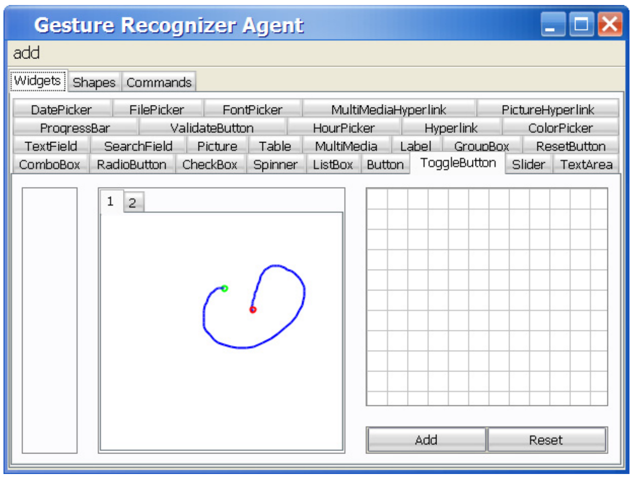


Figure 7. A graphical editor for a new object representation and a gesture recognition system where a new gesture replaces a predefined object (here, a gesture is drawn, added, and activated to represent a toggle button in a personalized way).

Multiple output formats. At any time, the tool produces UI specifications in terms of a User Interface Description Language (UIDL) instead of UI code, which is the prevalent approach of most tools [3,13,16,19]. As opposed to many tools where little or no portions of the sketch could be reused, our tool always maintains up-to-date UI specifications, including the description of custom objects. It is also possible to define the navigation between these objects in the same way to address the second and third dimensions of McCurdy [14] (Fig. 8). Whereas Fig. 5 graphically depicts the four levels of fidelity for expressing the presentation of a GUI, Fig. 7 shows how to define the corresponding dialog (or behavior) in terms of Event-Condition Action (ECA) systems. Predefined transitions exist, such as in DEMAIS [1], to express behaviors like: if the user clicks on this button, then this control is activated (or deactivated), if the value *v* is selected among the values of the widget *w*, then that information is displayed. Other behaviors include coupled widgets, like coupling a slider and the value representing a percentage (as in Fig. 1). These predefined transitions are specified in graphical direct manipulation. Other transitions are expressed in the ECA rule language inspired from the state chart mechanism in [8].

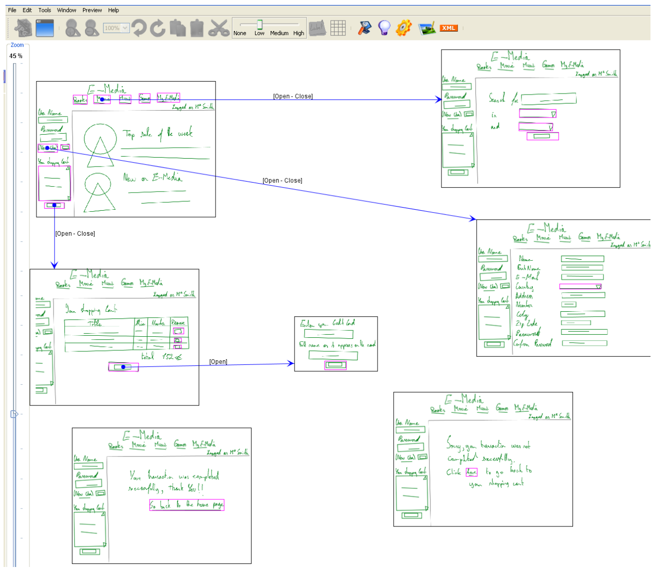


Figure 8. Definition of the dialog of UI objects.

Multi-platform UIs. By specifying project properties, the sketching tool enables designers to sketch UIs for a particular computing platform at a time (e.g., a desktop) or for several platforms in a coordinated way (e.g., a PDA, a laptop, and a desktop computer). It exports UI specifications in UIML (www.uiml.org), UsiXML (www.usixml.org – which is able to automatically generate code for XHTML, Java, and XUL). As opposed to some tools which are dedicated to a particular environment (e.g., Visual Basic in FreeForms [16], Java in JavaSketchIt2 [5]), MIXEDSKETCH is shipped with predefined profiles covering a wide range of different computing platforms, including but not limited to: mobile phone, smartphone, PocketPC, Tablet PC, laptop, and desktop. Each profile not only expresses constraints imposed by a particular platform (e.g., the screen resolution, a restricted widget set), but could also have a particular gesture data base for sketching those UI elements which are peculiar to this platform (e.g., a gesture associated to a histogram). Such specific UI elements are added by the gesture recognizer agent (Fig. 7). MIXEDSKETCH also renders a sketched UI for some particular Look & Feel of a computing platform (Fig. 9).

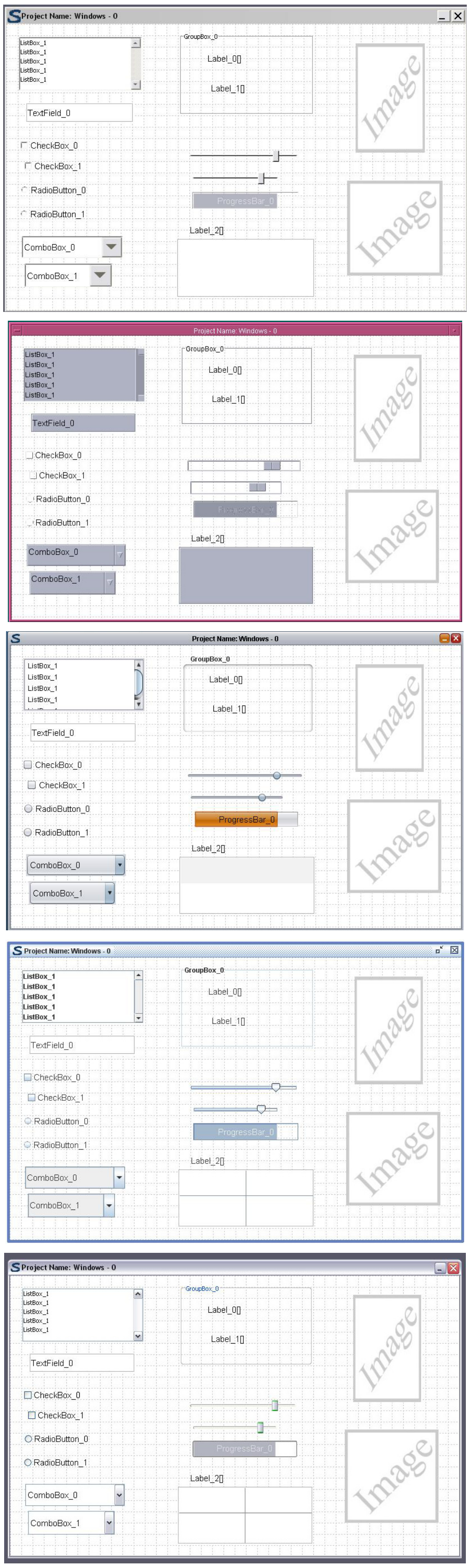


Figure 9. Rendering for different platforms.

PHYSICAL INTERFACE PROTOTYPING

This section describes how MIXEDSKETCH supports the prototyping life cycle of a TUI belonging to the physical world. In addition to the features presented in the previous section, other features are introduced to support prototyping TUIs belonging to the physical world.

Multiple input formats. At any time, a particular TUI object set can be specified in order to rely on the definitions available in the object set. If for instance, a TUI object set like Phidgets [7] offers predefined TUI objects, a representation can be defined for each such object that can be recognized either in the object recognition engine or in the gesture recognition engine. As long as the TUI objects of the set are defined according to the rules defined in the world modeling, they can be part of the sketching process whatever their purpose is. Examples are Pin & Play [6,23] and Voodoo [24]. Pin&Play [23] is a toolkit that allows developing physical interfaces by integrating software widgets and physical devices such as slider, toggle button, and potentiometer. Since the toggle button is not a standard element, is has been defined through a new custom gesture (Fig. 7), which could then be associated with the description of the genuine physical toggle button (such as a switch) belonging to the TUI object set. Fig. 10 respectively reproduces such a physical UI in Lo-Fi (a), Me-Fi (b), and Hi-Fi (c) with a smooth transition between these modes as previously described.

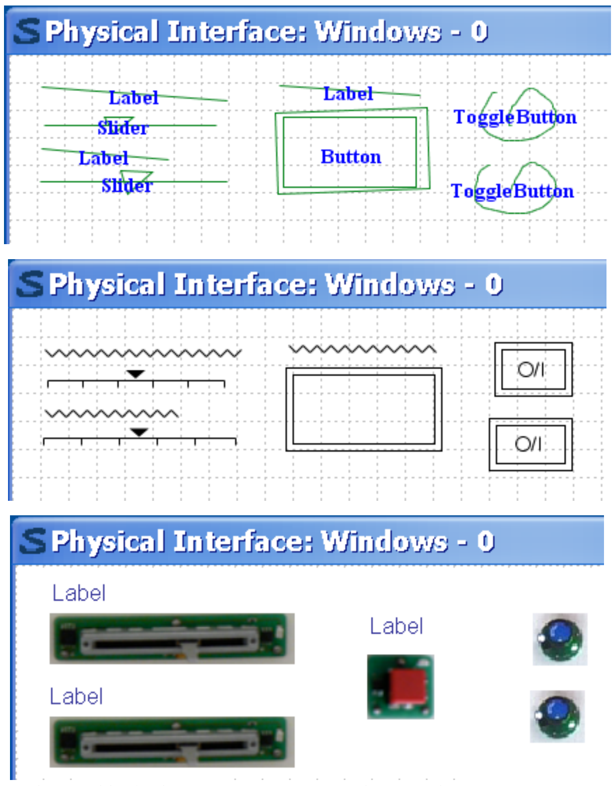

Physical Interface: Windows - 0
Label
Slider
Label
Slider
Label
Button
ToggleButton
ToggleButton
Physical Interface: Windows - 0
O/I
O/I
Physical Interface: Windows - 0
Label
Label
Label

Figure 10. A simple TUI sketched in Lo-Fi mode (a), then transformed into Me-Fi mode (b), and represented in terms of TUI elements in Hi-Fi mode (c).

Multiple output formats. Once a particular TUI has been specified, its internal model is automatically generated and augmented depending on information provided during the design stage. A mapping system exists that links each internal model object to the corresponding representation of the target output, like a description in Voodoo or in Pin&Play. Other TUI object sets could be equally supported provided they can define their design space in terms of objects and relationships as explained before. For the moment, the model is expressed in terms of a UIDL such as UsiXML (User Interface eXtensible Markup Language – http://www.usixml.org).

MIXED USER INTERFACE PROTOTYPING

Multiple rendering. In the two previous sections, we have seen that each UI corresponding to the digital or physical worlds can be sketched in exactly the same way: start at a particular level of fidelity (there is no need to start from Lo-Fi), move to any other level of fidelity as appropriate, and refine in the last Hi-Fi level. When the prototype has been validated by the end user, its specifications can be exported to a rendering engine. Two kinds of rendering engines exist:

1. An *interpreter* reads the generated specifications and graphically renders the corresponding UI in the environment. Fig. 1b shows a rendering engine in virtual reality so as to prototype UIs for ambient intelligence. If the TUI of Fig. 1a has been drawn, then a physical wall UI could be render (Fig. 1b – left) or a software counterpart (Fig. 1b- right). This is achieved in a very small amount of time by reinterpreting the UI specifications in the desired engine. Fig. 11 shows a screen shot of the VRML rendering engine where the same UI is projected on a wall, thus allowing the designer to explore alternative designs.

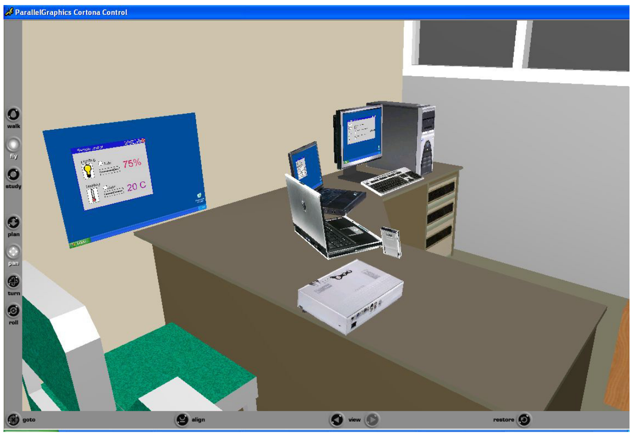


Figure 11. **A projected GUI corresponding to the GUI (Fig. 1).**

2. A *code compiler* reads the generated specifications and generate the code corresponding to the UI. Fig. 12 has transformed the TUI depicted in Fig. 10c into a GUI by relying on the mapping system. In this case, there is a mapping between GUI widgets and TUI objects. If such a mapping cannot be established, the initial object remains unchanged or is erased depending on preferences.

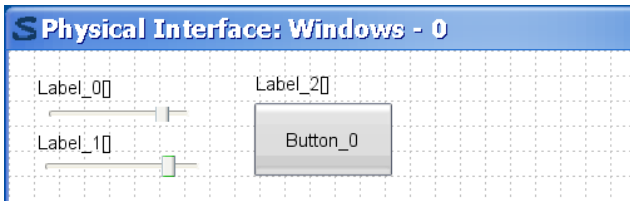


Figure 12. A simple GUI corresponding to the TUI of Fig. 10.

Therefore, not only it is possible to sketch a digital or physical UI at different levels of fidelity, but it is always possible to switch from one world to another and immediately see the corresponding UI in the other world. When no specific world is designated, a same UI being prototyped may involve GUI controls as well as TUI objects since their internal representation in the underlying model remains the same. In this way, a mixed UI could be similarly prototyped.

Transition from any world to another one. When a GUI is sketched in the digital world, MIXEDSKETCH is able to automatically generate a corresponding TUI (here a physical UI) in the physical world by searching the *closest behaviorally-equivalent element* in the target world. For instance, a slider widget (belonging to the digital world) can be mapped onto a physical slider (belonging to the physical world). The technique works also in the other direction: a TUI can be transformed into a GUI whose elements are found to be the closest behaviorally-equivalent to those found in the TUI.

In order to determine the closest behaviorally-equivalent element from one world to another, a UML class diagram has been realized that abstracts the most frequent elements into *abstract user interface objects* (Fig. 13). These objects are the core elements of a *abstract user interface model*, a model that describes canonically a user interface in terms of abstract interactors, containers and relationships in a way that is independent from the concrete interactors available on the target platforms. In practical terms this means that an abstract UI model is independent of any world, digital or physical. An *Abstract User Interface Object* is the root of the hierarchy of User Interface object. *AuiObjects* are the elements populating the AUI Model, they may be of two types:

AuiContainer is a container for grouping elements in order to define group of tasks that have to be presented together, in the same window or panel, for example. The container could contains *AuiInteractor* elements or another *AuiContainer* element. The relationship between *AuiContainer* and *AuiInteractor* is declared with a *AuiRelationship* object.

AuiInteractor is any individual element that populates an abstract container. It could be either a *DataInteractor* (i.e., an aggregation of the different types of elements that interacts with the user to present or obtain data which could be (input or output) and (PredefinedSet or UndefinedSet) or a *TriggerInteractor* (i.e., an aggregation of the different types of elements that triggers actions).

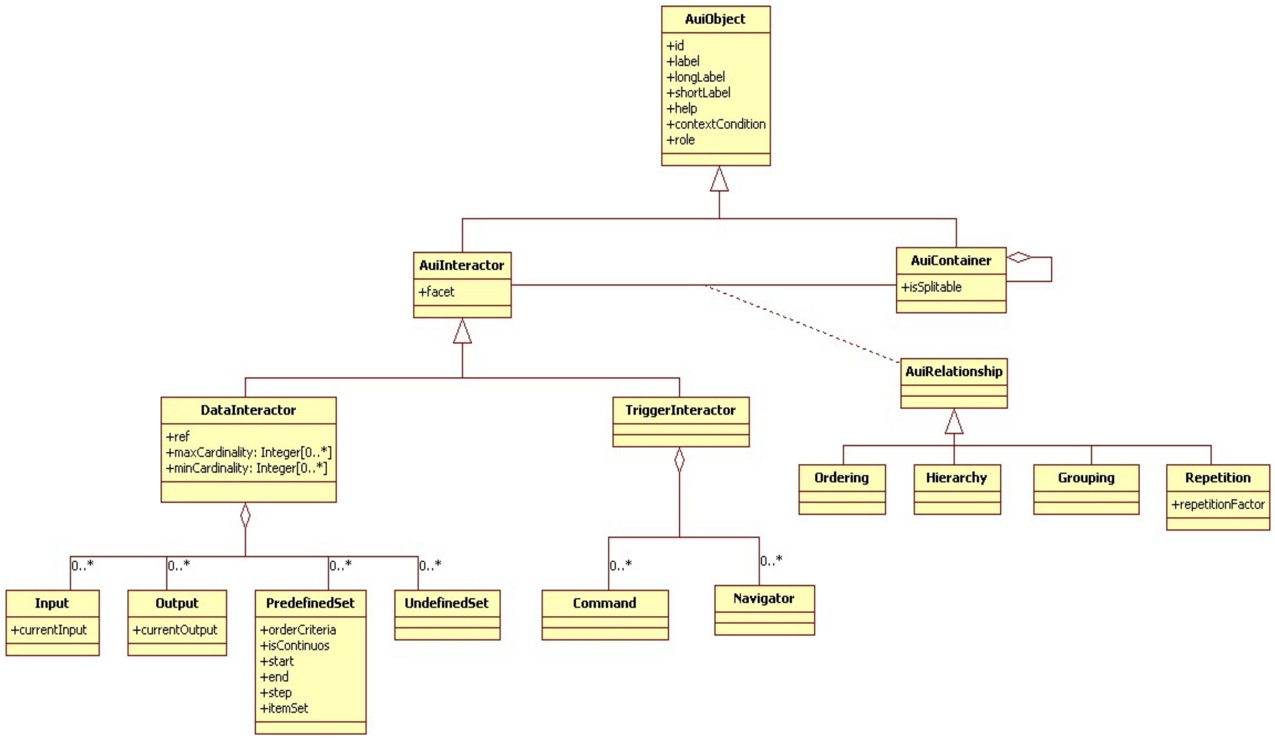


Figure 13. **The meta-model used for the abstract user interface characterization.**

An *AuiInteractor* cannot be simultaneously a *DataInteractor* and a *TriggerInteractor*. A *DataInteractor* could of course be attached to a particular behaviour depending on the data manipulated (i.e., entered, deleted, or modified).

Input is an element used for obtaining values from the user, *Output* is an element used for presenting values to the user, *PredefinedSet* describes the set of values valid for the element, *UndefinedSet* describes a set of open values (without predefined values). In this way, each *AuiInteractor* may hold one or several facets that are represented by the subclasses. The UML class diagram of Fig. 13 does not represent the constraint that each *DataInteractor* should have at least one facet represented by a sub-class.

Of course, an *AuiInteractor* may have until four facets: *Input*, *Output*, *PredefinedSet*, and *UndefinedSet*. Indeed, an interactor can serve for input only (no feedback), output only (only display), input/output (input that gives some system feedback) and still have a mixed domain of values: one part that is known at design-time (*PredefinedSet*) and another part that is unknown at run-time (*UnderfinedSet*).

TriggerInteractor is an aggregation of the different types of elements that triggers actions, while *CommandInteractor* is a element used for trigger a command. *NavigatorInteractor* is a trigger used for indicating a navigation to manage. Other classes and relationships are detailed at http://www.-anonymous.org. Here, we only detail the classes useful for searching the closest element.

Basically, any element, whether it is a GUI element belonging to the digital world or a TUI element belonging to the physical world, is linked to an abstraction expressed in the terms outlined in Fig. 13. For instance, a slider widget is characterized as a *DataInteractor* that is applicable for both *Input* and *Output*, based on a *PredefinedSet*. A physical slider is characterized exactly in the same way.

By definition, the closest behaviorally-equivalent element is any element that maximizes the matching in terms of the *AuiModel*. For this purpose, the system checks all the facets of existing elements in order to find any other element matching the same facets with any sub-set or super-set of values for them. The facets are examined first one by one (i.e., *Input, Output*, *PredefinedSet*, and *UndefinedSet*), then in combination of 2 (i.e., *Input* and *Output*, *Input* and *PredefinedSet*, *Input* and *UndefinedSet*), 3, or 4, if possible.

In the case of our slider, the answer is obvious since a perfect match exists. If this is not the case, the system searches the graph for the element that matches the best the input element by checking all facets and associated constraints. Therefore, one initial UI element of a source world can be mapped onto zero, one, or many UI element of the target world. When zero matching exists, the initial UI element is only replicated or erased depending on options set by the designer. When only one match exists, thus meaning a bijection, the mapping is straightforward: the corresponding element is selected and added in the UI model corresponding to the new UI of the new world. When many matches exist, they are ranked by level of similarity of behavior and the designer is then allowed to choose any of them. This metric consists of the sum of similarities for all properties hold in each facet of each element. The designer can specify a threshold beyond which this similarity is considered reached since, in general, no perfect match exist for all values for all properties. Fig. 14 shows some mappings between GUI elements and TUI elements maintained in ontology of MIXEDSKETCH.

This ontology has been created by exporting the UML class diagram of Fig. 13 in XMI, along with the mappings to respective elements. Although this file can be edited manually, it is the responsibility of the designer to maintain consistency between the mappings in order to avoid any mistake. MIXEDSKETCH then parses this ontology at run-time in order to browse the relationships between *AuiInteractors*. Actually, each UI element used in one world is abstracted into a corresponding *AuiInteractor*. Relationships between *AuiInteractors* are then exploited to find out the closest one.

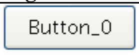
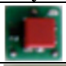
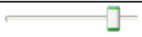
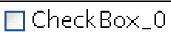
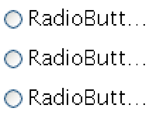
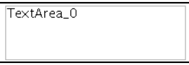
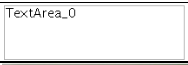
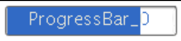
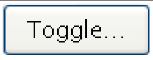
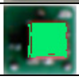
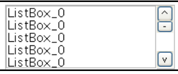
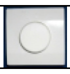
AuiInteractor	**Digital world**	**Physical world**
PushButton		
Slider		
CheckBox		
RadioButton		
TextArea		
ProgressBar		
ToggleBut-ton		
ListBox		
Rotator		

Figure 14. **Some mappings between GUI and TUI elements.**

Figure 15 shows the two families of input devices supported by MIXEDSKETCH: tactile screens (fig. 15a) and graphic tablets (fig. 15b). Graphic tablets impose less physical arm fatigue than tactile screens, but are intrinsically indirect input devices: the end user only sees the results on the main screen, not on the tablet.

CONCLUSION AND FUTURE WORK

Regarding to the MIXEDSKETCH tool, we demonstrated that it provides a user interface that is specifically de-signed to support hand gestures on a graphics tablet or any other interactive surface, such as electronic whiteboards. Whiteboards usually fall into two categories: Integrated back-projected electronic whiteboards and Modular front-projected electronic whiteboards. Gestures are pen strokes that are drawn by the user with a pen or mouse, and which are then interpreted by the program and replaced by a corresponding symbol (e.g., a widget representation, a diagram symbol).

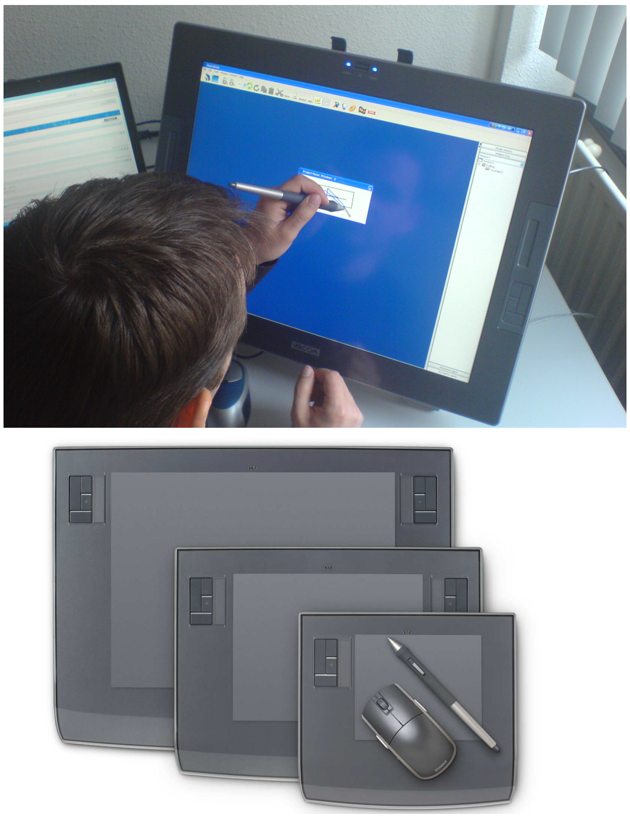


Figure 15. **Typical devices supported by MIXEDSKETCH.**

For instance, if one sketches the outline of a double box in MIXEDSKETCH, the tool will immediately interpret this as the gesture for a push button and replace this stroke with the representation corresponding to this widget. This representation may vary depending on the level of fidelity: none, low, medium, or high. The designer can then explore alternative designs by switching from one world to another. This switching is supported thanks to an ontology of abstract interactors, along with their relationships, that is maintained and exploited by MIXEDSKETCH. Currently, there are only two worlds (i.e., digital and physical).

Thanks to this ontology, one can imagine any new representation world that would introduce a new dimension with respect to the other ones. This new world should then be introduced in the ontology in terms of new element connected to an *AuiInteractor* along with its facets, its relationships, its representation in Hi-Fi (multiple representations of the same interactor are possible depending on some conditions to be stated), and the name of a corresponding element for export (e.g., the name of the objet in Voodoo), and its basic properties (e.g., dimensions, location, format).

In this way, we demonstrated that any person who is interested in sketching can actually draw a sketch of objects, regardless the world they are input (object recognition on multiple inputs), see them refined (multi-fidelity) and rendered (multi-rendering). Any object can be represented in the worlds it is supported (multi-world representation). Any object input in any world can initiate an object in any other corresponding world, provided that such a transition via the ontology exists (transition between worlds), and then be exported in a XML-compliant format for being recuperated in a toolkit. This transition ensures consistency between designs across worlds since the software naturally maintains this property thanks to the notion of behavior equivalence. When this property cannot be established, the closest one is selected instead. When objects belong to the digital world, the prototype covers main aspects of GUI prototyping, but for multi computing platforms with the ability to switch from one fidelity (e.g., the level at which the GUI was drawn) to another (e.g., Mi-Fi, Hi-Fi). When all objects belong to the physical world, the prototype covers main aspects of UI prototyping for ambient intelligence, home automation, virtual reality, etc. When both types of objects are present, the UI being prototyped is said to be mixed since it combines objects from both worlds.

REFERENCES

1. Bailey, B.P. and Konstan, J.A. Are informal tools better? Comparing DEMAIS, pencil and paper, and Authorware for early multimedia design. In *Proc. of CHI'03*, ACM Press, New York (2003), 313-320.
2. Ballagas, R., Ringer, M. Stone, M., and Borchers, J. iStuff: a physical user interface toolkit for ubiquitous computing environments. In *Proc. of CHI'2003*, ACM Press, New York (2003) 537-544.
3. Caetano, A., Goulart, N., Fonseca, M., and Jorge, J. JavaSketchIt: Issues in Sketching the Look of User Interfaces. In *Proc. of AAAI'02 Spring Symp. on Sketch Understanding*, AAAI Press, Menlo Park, 9-14.
4. Fitzmaurice, G., Ishii, H. and Buxton, W. Bricks: Laying the Foundations for Graspable User Interfaces. In *Proc. of CHI'95*, ACM Press, NY (1995) 442-449.
5. Fonseca, M., Jorge, J., and Garcia, F.M. Visual Syntax Analysis for Calligraphc Interfaces. In *Proc. of 13th Encontro Português de Computação Gráfica*, Universidade Trás-Os-Montes e Alto Douro (Vila Real, Oct. 2005).
6. Gellersen, H. Smart-Its: computers for artifacts in the physical world. *Com. of the ACM 48*(3), 2005, 66.
7. Greenberg, S and Fitchett, C. Phidgets: easy development of physical interfaces through physical widgets. In *Proc. of UIST'01*, ACM Press (2001), 209-218.
8. Hartmann, B., Klemmer, S. R., Bernstein, M., and Mehta, N. d.tools: Visually Prototyping Physical UIs through Statecharts. In *EA of UIST'2005*, ACM Press (2005).
9. Hong, J.I. and Landay, J.A. Satin: a toolkit for informal ink-based applications. In *Proc. of UIST'00*, 63-72.
10. Klemmer, S.R., Integrating Physical and Digital Interactions. *IEEE Computer* (2005) 111-113.
11. Landay, J.A. and Myers, B.A. Interactive Sketching for the Early Stages of User Interface Design. In *Proc. of CHI'95*, ACM Press, New York (1995), 43-50.
12. Lin, J., Thomsen, M., and Landay, J.A. A visual language for sketching large and complex interactive designs. In *Proc. of CHI'02*, ACM Press, 307-314.
13. Lin, J. and Landay, J.A. Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces. In *Proc. of CHI'2008*, pp. 1313-1322.
14. 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 *Proc. of CHI'06*, ACM Press, New York (2006), 1233-1242.
15. Pedersen, E. Sokoler, T. Nelson L. PaperButtons: expanding a tangible user interface. In *Proc. of DIS'00*, ACM Press, New York (2000) 216-223.
16. Plimmer, B.E. and Apperley, M. Interacting with Sketched Interface Designs: an Evaluation Study. In *Extended Proc. of CHI'04*, ACM Press, pp. 1337-1340.
17. Rudd, J., Stern, K., and Isensee, S. Low vs. high-fidelity prototyping debate. *Interactions 3*(1), 1996, 76-85.
18. Sefelin, R., Tscheligi, M., and Giller, V. Paper Prototyping – What is it Good for? A Comparison of Paper-and Computer-based Prototyping. In *Proc. of CHI'03*, ACM Press, New York (2003), 778-779.
19. Signer, B. and Norrie, M.C. PaperPoint: A Paper-Based Presentation and Interactive Paper Prototyping Tool. In *Proc. of TEI'07*, ACM Press, New York (2007) 57-64.
20. Snyder, C. *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces*. Series in Interactive Technologies, Morgan Kaufmann, 2002.
21. Spiessl, W., Villar, N., Gellersen, H., and Schmidt, A. VoodooFlash: authoring across physical and digital form. In *Proc. of TEI'2007*, ACM Press (2007), 97-100.
22. Tohidi, M., Buxton, W., Baecker, R., and Sellen, A. User Sketches: a Quick, Inexpensive, and Effective Way to Elicit more Reflective User Feedback. In *Proc. of NordiCHI'06*, ACM Press, New York, 105-114.
23. Van Laerhoven, K., Villar, N., Schmidt, A., Gellersen, H., Håkansson, M., and Holmquist L.E. Pin&Play: The Surface as Network Medium. *IEEE Communications Magazine 41*(4), April 2003, 90-96.
24. Villar, N., Gilleade, K., Raymundy-Ellis, D., and Gellersen, H. The VoodooIO Gaming Kit: A Real-Time Adaptable Gaming Controller. In *Proc. of ACE'06*, ACM Press, New York (2006).
25. Villar, N. and Gellersen, H. A Malleable Control Structure for Softwired User Interfaces. In *Proc. of TEI'07*, ACM Press, New York (2007) 49-56.
26. Virzi, R.A., Sokolov, J.L., and Karis, D. Usability problem identification using both low- and high-fidelity prototypes. In *Proc. of CHI'96*, ACM Press, 236-243.
27. Walker, M., Takayama, L., and Landay, J. High-fidelity or Low-fidelity, Paper or Computer medium? In *Proc. of HFES'02*, HFES, Santa Monica (2002), 661-665.