Revista Romana de Interactiune Om-Calculator 9 (4) 2016, 305-333 © MatrixRom

The UsiSketch Software Architecture

Jorge Luis Pérez Medina[1]

[1] Université catholique de Louvain, Louvain School of Management Research Institute
Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
E-mail:jorge.perezmedina@uclouvain.be

Abstract. In previous work, we proposed a method to facilitate the tabletop collaborative prototyping of model-based user interfaces in early steps of the design process when multiple stakeholders have only a vague goal in mind of what should be produced. We also developed UsiSketch, a Java-based software supporting our modeling sketching method. In this paper, we present the main functionalities and the software architecture of UsiSketch.

Keywords: Sketching, Collaborative Prototyping, Graphical User Interface, Design Tools and Techniques.

1. Introduction

In previous work, we presented UsiSketch (Pérez-Medina, 2016), a method for modeling interfaces collaboratively for the prototyping of user interfaces. The proposed method is composed of three main phases, namely: pre-production, production, and execution phases. The pre-production phase aims at defining the underlying grammar of the gestures to be made during the design session and at training the algorithms for enhancing their recognition capabilities; the production phase regards the recognition of performed gestures on a large surface and the creation of an XML file as output; the execution phase aims at executing a simulation of the designed UI.

The motivation and main functionality of the method were discussed. The main contribution presented was a novel strategy to handle sketching on very large interaction surfaces.

This work will expand on this idea, but while the focus of the previous work was dedicated to the new recognition algorithm that accommodates very large surfaces and model-based design of user interfaces with collaboration, this work focuses on the description of the software architecture and the tool called UsiSketch that incorporates the

aforementioned techniques. The description of the main functionalities of the tool and its main benefits and shortcomings are also detailled. The software architecture supports only two physical configurations of a Collaborative User Centered Design method (Pérez-Medina et al., 2016) based on sketching to support prototyping along a Software Development process. Based on the spatio-temporal relationships among designers and users the physical configurations consider only co-located synchronous collaborations of the different users (or roles, which contains both designers and users) and devices. Designers and final users can collaborate at the same time and in the same place using a single device. The screen size, of the device configurations, supported by the architecture can be small, medium, large desktop, extra large desktop and wall screen.

The rest of this paper is structured as follows: Section 2 presents an analysis of related work. Section 3 describes a set of technical requirements identified for UsiSketch and provides an analysis of the architecture used to develop the software in Section 4. UsiSketch tool is described in Section 5. Section 6 gives some benefits and shortcomings. Finally, section 7 presents conclusions and future work.

2. Related work

The related work to UsiSketch could be divided into two parts: research related to user interface prototyping by sketching and research related to collaborative design. This section attempts to address both.

Ambler (2009) found that a prototyping tool aims to facilitate prototyping of user interface as an iterative analytical technique in which users are actively involved in the design phase. (Vanderdonckt & Coyette, 2007) express that a user interface prototype has many uses: exploring the problems posed by the system to design, identifying the constraints of the system to design, developing possible solutions for the system to design, communicating the different possible UIs for the system and laying the foundations from which the system can be built. During prototyping, the designer must produce a close representation of the final software to validate its ergonomics, the typical scenarios and/or the frequent use, among others. The importance of this step is not the drawing of the final interface, but a faithful representation sufficient to take out the appropriate conclusions.

The concept of fidelity of a prototype is the object of several research projects, for example, (Landay & Myers, 1995; Newman & al., 2003; Petrie & Schneider, 2006). SILK (Landay & Myers, 1995) proposes a natural mechanism of conception where the designer freely design draws the interface being designed as a free hand graphic design tool. Prototyping a UI to a low fidelity level enables discovering many problems at a higher level. The fidelity level express the similarity between the gesture representation of the prototyped interface and the interface itself (Vanderdonckt & Coyette, 2007).

In (Vanderdonckt & Coyette, 2007), the authors afirm that mixed-fidelity prototyping consists of mixing simultaneously or not various fidelity levels within the same user interface prototype. Multi-fidelity prototpying incorporates mixed-fidelity prototyping in that it enables the designer to express any interaction object in multiple fidelity levels and ensure a smooth and dynamic transition between these levels at design-time. The multi-fidelity approach allows integrating user interface elements delivered in one or many fidelity levels in order to build a user interface prototype which evolves as the interactive application development life cycle is progressing. Prototyping to high fidelity level may follow a prototyping to a lower fidelity level as raised above, but not necessarily.

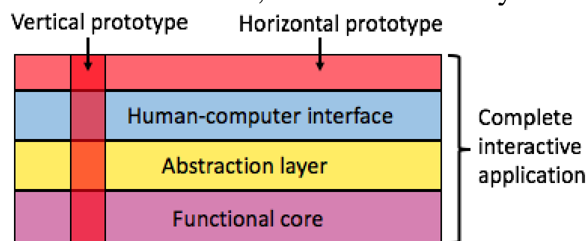

Figure 1. Vertical and horizontal Prototype. Adapted to (Vanderdonck & Coyette, 2007).

The Nielsen's Reference (1993), presented in Figure 1, distinguishes two types of prototypes depending on the level of interactivity covered: vertical and horizontal. In (Vanderdonckt & Coyette, 2007), the authors suggest that prototyping can be performed in three main ways: the so-called human-computer interface, the abstraction layer of the application and the functional core comprising the semantic functions of application (see Figure 1). The type of prototyping is called horizontal when designers want to develop a maximum of functionalities of the application through the interface. The interface is then prototyped to a low or medium fidelity level

in order to verify that all the functionalities are well identified and covered. Consequently, the prototyping is not deep since only the main aspects (e.g. the presentation) take place over the detailed aspects (e.g. the really implemented functionalities). When the first horizontal layer is completed, for instance: through a validation performed by final users, the prototyping activity can spread to the lower levels (see Figure 1). In this process, the interface is first completed, the abstraction layer is then initiated, next, the semantic functions of the functional core. This corresponds to a situation where the interface is the primary element in prototype before any other layer. Once it is stabilized, the associated behavior can be developed without fear of seeing it changed too quickly.

Tahuti proposed by (Hammond & Davis, 2002) is a multi-stroke sketch recognition environment for class diagrams in UML where users can sketch the diagrams on a table or whiteboard in the same way they would on paper and the sketches are interpreted by the computer.

Tablaction created by (Kim et al., 2010) supports collaborative brainstorming process which supports simultaneous stylus and multi-touch finger inputs. This work also discusses some ideas regarding the limitations faced by participants in a brainstorming meeting. We agree that people cannot equally participate in the brainstorming and we argue that the use of large screen could be an alternative solution. Additionally, the authors present a paper prototype test performed with a specially designed stylus for it. The authors expect the interaction design with multi-touch and stylus capabilities on tablets to accelerate idea expressions in a brainstorming meeting.

Dazzle proposed by (Oehlberg et al., 2012) associates the action of showing information on the shared display with granting the rest of the team access to that information: showing is sharing. Dazzle also records a history of shown files. Team members can annotate this log using cross-platform synchronized clients.

(Safin & Leclercq, 2009) evaluate the opportunities and constraints linked to the technological transfer of a sketch-based distant collaborative environment, from academy to industry. The paper relates the concepts of the sketch-based collaboration, describes the Distant Collaborative Design Studio and proposes a methodology to assess the utility and usability of the system in two different companies. The results and conclusions show the issues linked to the implementation of such sketch-based collaborative environment in professional contexts.

WebSurface produced by (Tuddenham et al., 2009) proposes an alternative approach to drawing ideas in a collaborative way from tabletop interfaces. It explores an alternative approach to the problem of collaborative Web browsing by applying recent techniques from tabletop interfaces: large horizontal collaborative surfaces. The findings suggest that a tabletop approach for collaborative web browsing can help address limitations of conventional tools, and presents beneficial affordances for information layout.

Another system to support UI design on tangible surfaces is shown in the Designer's Outpost proposed by (Klemmer et al., 2001). It found that pens, paper, walls, and tables were often used to explain, develop, and communicate ideas during the early phases of design. These wall-scale paper-based design practices inspired The Designers' Outpost, a tangible user interface that combines the affordances of paper and large physical workspaces with the advantages of electronic media to support information design. With the tools, users collaboratively author web site information architectures on an electronic whiteboard using physical media (Post-it notes and images), structuring and annotating that information with electronic pens. This interaction is enabled by a touch-sensitive SMART Board augmented by a robust computer vision system, employing a rear-mounted video camera to capture movement and a front-mounted high-resolution camera to capture ink.

(Cherubini et al., 2007) presents findings of an exploratory study of how and why developers draw their code. The study focused on the social practices around diagrams and visualizations. The authors found that diagrams play largely a supportive role in software design and that drawings are often ephemeral because of the labor involved in translating them into more permanent forms. These findings and others provide useful insights into the design of a wide array of software-visualization tools as well as into the use of diagrams in design work in general. Among the results, we find that, in most cases, informal notation was used to support face-to-face communication and that current tools were not capable of supporting this need because they did not help developers externalize their mental models of code. Instead, developers reported that the level of abstraction differs with every conversation and even within a conversation.

Different tools for modeling sketches and prototyping are proposed in the literature. We will not produce a comparison of these tools as it would be

beyond the scope of this paper. However, we consider necessary to give a list of some electronic tools to support sketching activities, especially Gambit by (Sangiorgi & Vanderdockt, 2012) a cross-platform tool conceived to support a collaborative User Centered Design (UCD) method described in (Norman & Draper, 1986) and (Sangiorgi et al., 2012) to foster creativity and discuss design ideas studied in van der Lugt (2002). The sketching tool list related to UsiSketch is: JavaSketchIt (Caetano et al., 2002), Damask by (Lin & Landay, 2002), SketchiXML proposed by (Coyette et al., 2004), Denim by (Lin & Landay, 2008), Sketch API by (Sangiorgi & Barbosa, 2010), Sketchify produced by (Obrenovic & Martens, 2011), FlexiSketch by (West et al., 2013) , Rapido by Mitra (2015) and FlexiSketch TEAM by (Wuest et al., 2015). FlexiSketch TEAM is a solution for collaborative, model-based sketching of free-form diagrams. It allows multiple users using their own tables to work simultaneously on the same model sketch and use lightweight metamodeling mechanics to collaboratively define custom notations on the fly. The similarity of FlexiSketch TEAM with our tool is that users can define the syntax for sketched symbols and links. However, the tools do not support distributed collaboration with Multi-level prototyping, the exportation to a generic user interface description language and the independent execution of the user interface. The purpose of SketchiXML is the same as that of UsiSketch: to sketch and simulate the user interface. However, the tools supports a relatively low fidelity. The drawn widgets are not all the same size and they are not aligned. This represents a great limitation when working on large surface.

3. Key requirements

We argue that a software that supports sketching and simulation on multiple device and platforms would enrich the collaboration of designers and final users during requirements capture and analysis. One important issue with sketch-based systems for prototyping and simulation of user interfaces is that they must adapt the recognition of gestures on small and large screens.

For define the requirements of UsiSketch an extensive literature review was conducted in the areas of : (1) prototyping, specially with paper-pencil; (2) interfaces edition; (3) interface description languages; (4) hardware and

pointing material.

Naturalness, intuitiveness, simulability, completeness, low-high fidelity levels, exportability, collaborativity, short learning time, short production time are some of the attributes desired by UsiSketch as a prototyping tools. Today, prototyping activity is still on paper and many designers still consider paper prototyping an efficient method (Ambler, 2007). Indeed, nothing is more natural, when talking about prototyping, to take a sheet of paper and shetch what is expected to see on the screen. It is a flexible, intuitive and accessible method. Our interest is that the computer can analyze and simulate the beahvoir of paper prototype allowing the designer to sketch the user interfaces as easily as on paper.

Although UsiSketch can be used with a traditional pointing device (mouse, track-pad), this tool must mainly designed for use with a pencil-type pointing tool.

In the context of a Collaborative User Centered Design method based on sketching we argue that a user working alone will preferably use a small or desktop surface, while a group of people will be more comfortable with a wall screen because the screen is projected, therefore more easily readable by several people. Additionally, working directly on the screen give a natural and better immersion. This kind of device is very interesting to favor the intuitiveness of UsiSketch.

Usually, at the end of the prototyping step two types of prototypes are produced: Disposable prototypes (the prototype is validated, thrown and re-developed) or non-disposable (all or part of the prototype are recover for further development). Unlike pure prototyping tools, interface editors are not meant to create / simulate a prototype, but to define the final interface used in a software. In this respect, their export capacity is generally very good, as is their level of fidelity. They are also software of type WYSIWYG, but they often offer the possibility of editing the source code directly (if it is human-readable).

The techniques used by the interface editors and those of the prototyping tools are generally similar. The fundamental difference between these two classes of software is their mission: one grouping together software that helps in pure prototyping, while the other serves to create interfaces that will actually be used. We are interested in a prototyping tool were the prototype can be created, modified and once validated reused in later stages of the development process. The reuse of an UI prototype requires the

consideration of an interface description language (García et al., 2009). A systematic comparison of these language is outside the scope of this paper. The interested reader may refer to (García et al., 2009) for a more comprenhensive comparison. However, our focus is on UsiXML validated by the W3C (Consortium usiXML, 2007) which gives a good visibility, guarantees quality and a good standardization of the language.

We rely on the VOLERE model[11] for the discovery and representation of the requirements since it provides a set of resources for eliciting and specifying requirements allowing also to improve requirements specifications. We started by defining the main requirements, then we have gradually added the smaller requirements. Each requirement was classified into a list according to a importance level.

Based on the literature review and examining related work and especially in (Coyette et al., 2004; West et al., 2015a; West et al., 2015b), we identified a total of 42 main requirements for modeling sketches on very large surfaces. The beneficiaries of these requirements are: Designers, Testers and Developers. These requirements are grouped in Recognition (8), Drawing-Rendering (9), Prototyping (4), Data (4), Simulation (4), Ergonomics/Usability (6), and Architecture (7). The detail of these requirements is beyond the scope of this paper, however, the requirements are briefly described in the appendix sections. Through the next section we will introduce UsiSketch which was developed according the list of requirements.

4. UsiSketch software architecture

UsiSketch is an Eclipse plug-in developed in Java which supports multiple computing platforms. It integrates the new algorithm, based on the described method that recognizes UI sketching on very large surfaces (Pérez-Medina, 2016). Figure 2 shows the principal components of the UsiSketch. The general architecture is inspired by Model-View-Controller. The tool uses the following libraries:

1. Eclipse Sketch developed by (Sangiorgi et al., 2010) contains the shape recognition algorithms used in usiSketch.
2. UsiXML conceived by the Consortium usiXML (2007) contains all Java classes to parsing and exports the forms to the User Interface

[11] Volere Requirements home page. www.volere.co.uk.

eXtensible Markup Language (usiXML), an XML-compliant markup language that describes the UI for multiple context of use as character User Interfaces (CUIs), Graphical User Interfaces (GUIs), Auditory User Interfaces, and Multimodal User Interfaces.

3. Castor Project[12] provides the connection between XML and Java. It is used in conjunction with the UsiXML library for parsing and exporting a UsiXML file. Castor depends on both Commons-Logging and JDOM libraries.
4. Commons-logging[13] is in charge of the logging of all events when exporting to UsiXML.
5. JDOM[14] is required by UsiXML. It provides a complete, Java-based solution for accessing, manipulating, and outputting XML data from Java code.

We defined a utilization as a direct call from one module to another. An event is a message sent from a module and captured by another. Utilization and event of modules Config, Util and Events are not described.

4.1 The model package

The Model view includes the package: Graphics and Actions. The following sections describe these packages.

4.1.1 Graphics

*Graphic*s which contains all graphical elements used by the tool, as well as the structures for storage. The elements are not the components displayed on the screen, but the data required for their design (position, size, dots, ...).

There are four categories of objects, the first three contain the graphic

---

[12] Castor Project. Castor is an open source data binding framework for Java[tm]. It is the shortest path between Java objects, XML documents and relational tables. Castor provides Java-to-XML binding, Java-to-SQL persistence, and more. (http://castor-data-binding.github.io/castor/).

[13] Commons-logging. The Apache Commons Logging (JCL) provides a Log interface that is intended to be both light-weight and an independent abstraction of other logging toolkits. It provides the middleware/tooling developer with a simple logging abstraction that allows the user (application developer) to plug in a specific logging implementation. (https://commons.apache.org/proper/ commons-logging/).

[14] Jdom is a Java representation of an XML document. It provides a complete, Java-based solution for accessing, manipulating, and outputting XML data from Java code. (http://www.jdom.org/).

elements inheriting of the *Graphics* class. The fourth category corresponds to the elementary storage structures:

1. A *DotSet* elements represent a hand drawing. This element is a set of points connected in the form of a curve.
2. The *VectorialShape* element represents a vector shape. Several classes inherit of *VectorialShape* depending on the type of form defined. The *VectorialShape* allows manipulating a geometric form and converting it in a smoothed representation.
3. *Widget* represents a Widget structure. The widget is the constitutive element of a Graphical User Interface (GUI). Several classes inherit from *Widget*. It can be a button, a text, a text field, a listBox, etc. The purpose of this class is to inform the user, or allow him to interact with the system used.
4. The *GraphicsContainer* is the structure of storage for elements of type *VectorialShape* and *Widgets*. The *AnnotationsContainer* is the structure of storage for elements of type *DotSet* used as annotations. *Windows* represents an element containing its *GraphicsContainer*, its *AnnotationContainer* and its *HistoryManager*.

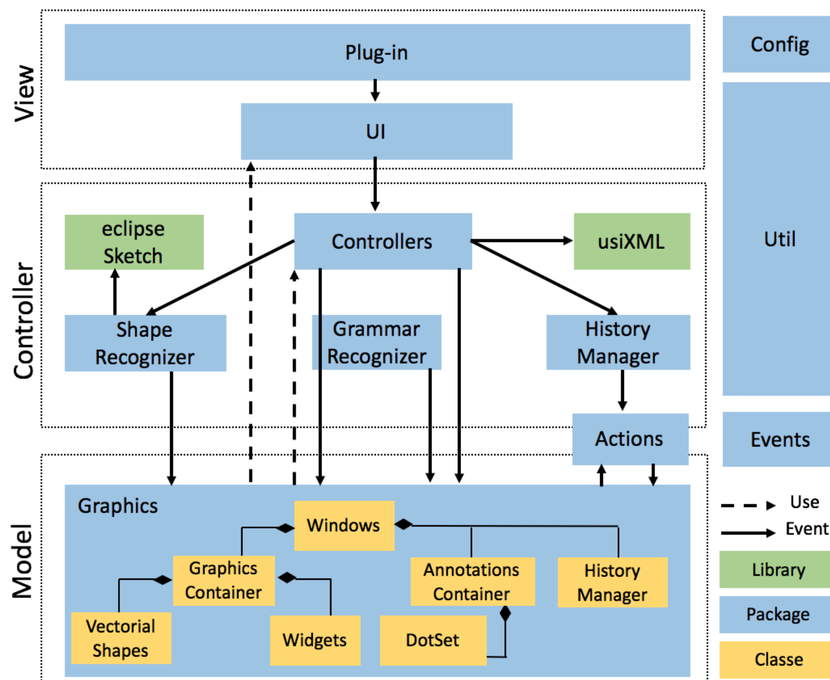

Figure 2. Execution phase for Modeling Sketches.

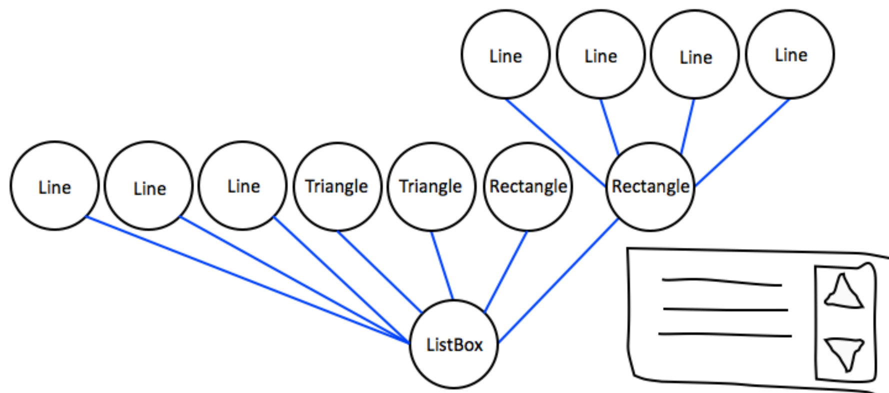

Figure 3. Data structure of graphic elements for the listBox widget.

To store the graphic elements other than *DotsSet*, a specialized structure was developed. It is designed to fulfill the requirement corresponding to the management of several fidelity levels, changeable at any time. Figure 3 shows a graphical idea of the data structure. It corresponds to an inverted tree whose nodes are graphic elements. In particular the leaves are *VectorialShapes* recognized by the recognition engine. The addition of drawn shapes is always performed in the leaves.

The structure also has a compose method which performs the composition of forms in another form. The compound form then becomes the parent of the nodes that comprise it. This method is only called by the composition engine shapes.

When rendering, the path of this structure behaves differently depending on the level of fidelity. For instance:

1. At the level of “fidelity widgets” and above: only root nodes are searched.
2. At the level of “fidelity forms”: child nodes are drawn if and only if the parent node is a *Widget*. In this case the parent node is not drawn.
3. At the level of “fidelity drawing”: only the leaves are drawn, in its drawn shape really.

4.1.2 Actions

The “Actions” package groups all undoable actions. This module is at the border of the model and controller view. The class contained in this package

defines the methods undo() and redo() which implement the action to perform. They are also used to manage the historical design, drawing as well as when saving the file. The classes contained in this package are created by HistoryManager, and the objects are stored in a log implemented by the HistoricList class contained in this module.

4.2 The view package

The *View* manipulates the visual aspects. It contains the interfaces implemented in eclipse and the connections made as a plug-ing. The *Plug-in package* contains all the classes in charge of connecting UsiSketch to eclipse. This module is also in charge of loading the views, opening files, etc. The *UI package* contains all views of the application. Each view allows managing the events that occur in it, and transmits the information to the controller concerned, depending on the event. The views defined are:

1. *WindowsEditor* is in charge of managing the design of forms and widgets.
2. *PreviewFrame* is in charge of the screen simulation.
3. *HistoryView* is in charge of listing the history of design, and allows the user to return to a previous state.
4. *GrammarView*. In this view the designer can specify the grammars required by the composition of widgets.

4.3 The controller package

The Controller view is in charge of the functional operations made in UsiSketch. There may be recognition processing, combination, user actions, etc. The main classes of the package Controllers are:

1. *SketchingController* which manages any action done in a drawing. It may include the addition of a sketch, and a suppression or selection command.
2. *ActionController* manages the actions carried out in an interface. More concretely, it deals with control buttons in the windows view.
3. *HistoryController* manages the actions performed in the *HistoryView*.
4. The *ShapeRecognizer* package implements a thread which recognizes the designed geometric forms and converts them into a smoothed form. Its package recognizes a form based on a set of points implemented by the class *DotSet*.

5. The *GrammarRecognizer* combines several simple forms in a complex form or widget, according to pre-established rules.
6. The *HistoryManager* maintains a desing of history and draws a prototype at any time.

4.4 Others packages

The *Config package* includes the variables of the plug-in. These variables can be used to change the behavior or appearance of the software. The *Util package* includes all classes used around the code. Finally, the *Events package* implements the design pattern Observer-Observable. It manipulates *EventTrigger* and *EventListener classes*. The event system is used to notify changes of the lower layers of software to the upper layers. For example, when a shape is added to the list of forms in a window, the container sends an event to all the views and controllers involved. The controller will then ask the *GrammarRecognizer package* to seek the new correspondences, hoping to identify a new widget.

5. UsiSketch tool

In order to give the reader a good understanding of UsiSketch, this section proposes the main features of the tool. UsiSketch is a prototyping software based on the sketch, enabling fast prototype, intuitive, a low level of loyalty, and reusable in the later stages of a software development life cycle.

Figure 4 shows the main screen of UsiSketch. UsiSketch's graphical user interface is decomposed into three parts. It has a tool bar at the top of the window that gives access to most of the features. The left side of the window gives access to a list of files created by a UsiSketch project. The last major component of the window is the workspace.

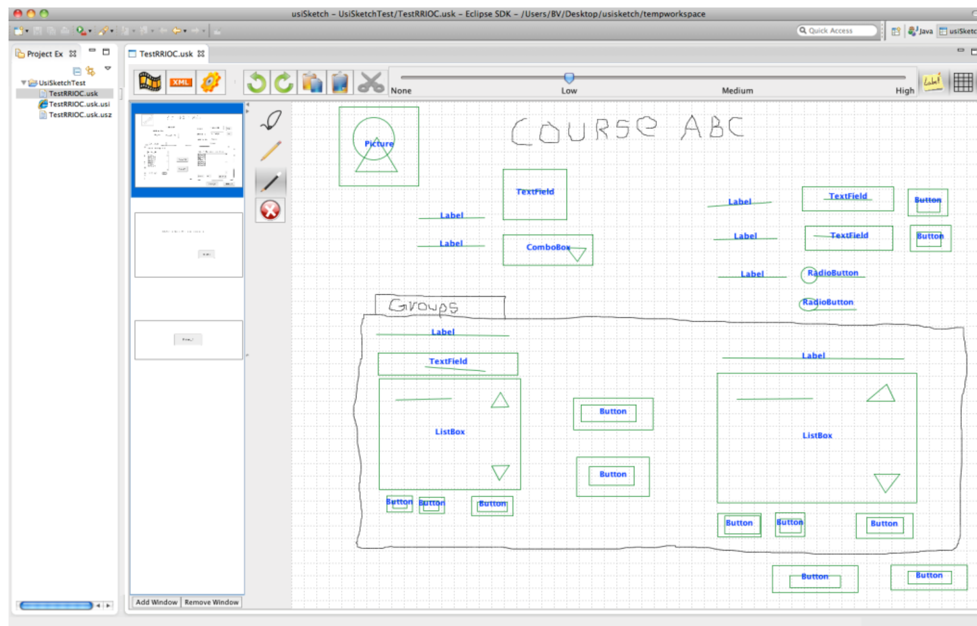

Figure 4. UsiSketch's graphical user interface.

The workspace is composed of a tool bar with the commands such as create a new UI, generate the XML representation of the UI, undo, redo, copy, paste, and change the fidelity level. Each of these commands can be associated to one or several gestures. The fidelity slider allows the designer to switch from fidelity levels to another just by moving the cursor. The "none" rendering consists of leaving the drawing as it is without any kind of beautification. The "low" level proposes a smoother representation as all the vectorial shapes composing the widget are replaced by a smoother representation. The "medium" level is a smoother representation than the previous one. Its representation is made independently of the shapes that were used to build the widget. Even if the representation of the textfield was composed of two intersecting circles the representation remains the same. Finally, the "high" level transforms the recognized widget by its corresponding widget in language Java/Swing. A second part of the workspace is the window selection area. It contains all the windows created in a UsiSketch project.

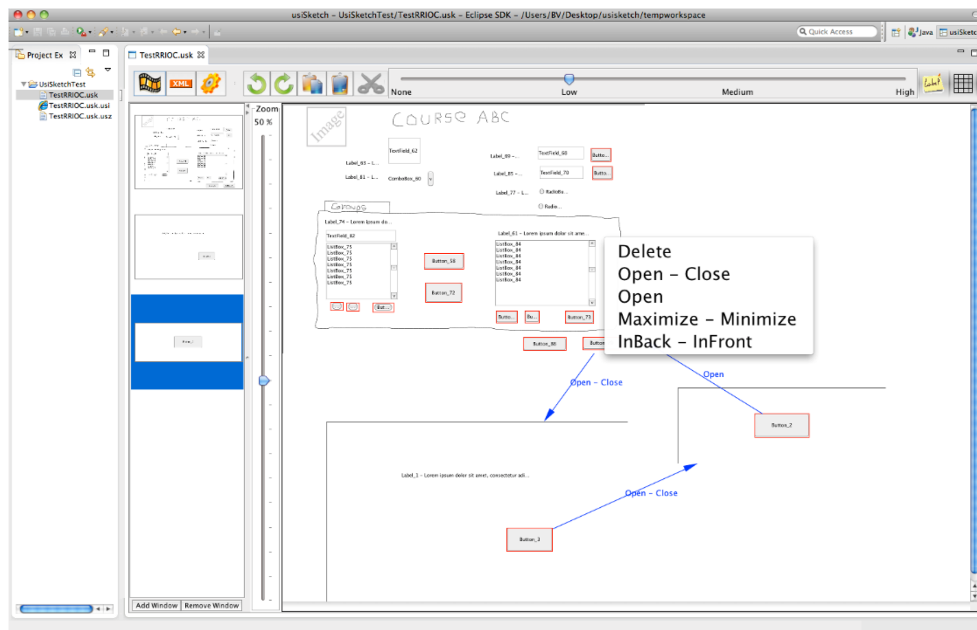

Figure 5. The navigation editor of UsiSketch.

Designers can create/remove a selected window, select a window and edit it. The drawing canvas is the key component of UsiSketch where the designers will draw the user interface. When designers desire to add a new representation to the UI, they just need to draw the representation where needed. For each widget, a set of representations is predefined and can be extended according to the designers' wishes. Each of these representations consists of a group of multi-stroke shapes, widgets or gestures and a list of contraints. For instance, we can observe on the top of the drawing canvas area that a picture is decomposed into three elements: a circle, a triangle, and a rectangle. The constraints specify that the circle and triangle must be inside the rectangle and the circle must be on the top of the triangle.

All these functionalities presented above permit proceeding to the design as naturally as paper sketch. UsiSketch also incorporates advanced editing functions to test the interaction of the user interfaces. Once the set of user interfaces composing the project is completed, designers have the possibility to sketch the relations between those screens. Figure 5 illustrates the relations between screens. Based on these relations, UsiSketch allows switching to a run mode where the end-user will have the opportunity to test a running prototype without third-party.

The run mode is part of the noticeable functionalities we integrated in this tool. As stated earlier, integrating an expressive scenario editor was very important, but we consider that testing this scenario easily is also crucial. The run mode requires a designer to play the computer and move the window accordingly to the user actions. So, UsiSketch incorporates a navigation editor. The editor uses all information provided by the final user and builds a run mode based on the sketches or the windows interpreted in java. UsiSketch also produces an output that is general and context independent. The output is based on user interfaces description languages such as UsiXML and UIML.

The design history is another evolution that was integrated in UsiSketch. When prototyping, a designer will try to explore many desings and evolve very fast. Looking back to the previous steps can be very useful. In UsiSketch, the designer can have a preview of the previous step and if needed go back to this step at any moment.

5.1 Assessment requirements

UsiSketch is under development. Some requirements are not fully completed. This section discusses the progress in the development.

The major features still missing are the ability to import an image and convert it into UsiSketch file, and the recognition tests. In addition, the composition rules of a widget are not editable via an interface yet.

In relation to the Drawing/Rendering, all the functionalities provided during the requirements are implemented, except for the selection and displacement of previously drawn shapes. However, the selection functionality is under development.

We find that the absence of a list of available widgets and their grammar could bring problems in learning gestures. This is a significant barrier to offer an intuitive software with fast handling. However, this problem will be partially solved when the grammar editing window is be available. However, we think that it will not be enough. We consider that a direct visual display of the window (without possibility of change) is needed in addition to the grammar editing view.

It would take a direct visual display of the window (no possibility of change) in addition to the grammar editing view. This list will be accessible at any time (e.g. through a button) and would display all existing widgets and their grammar. This feature also requires a relatively large effort to

implementation based on a generic grammar because we need to build a representative image.

Another aspect that we must consider is the navigation. The navigation is functional but still very perfectible. Indeed, the only possible actions are on the windows when clicking on a clickable widget (e.g. a button). Moreover, the only recognized events is the click on an item.

A UsiSketch project can be saved as a .usk file. Exporting a UsiSketch project into UsiXML file is also implemented. However, navigation rules from one page to the other are not exported yet.

Actually, an interface defined in UsiSketch must be simulated summary. However, this functionality can be enhanced. The only proposed dynamic behavior is the click of a button. It is not possible to change the text of a field, check a box or other dynamic behavior. Although it is not vital to use the software, Improving this functionality would be appreciated.

It is interesting to see that for writing text, some volunteers have preferred to use the annotation mode rather than draw lines representing labels, but only for static text. Similarly, they also added via the button text annotation. We believe that it is better to use the annotations mode instead of drawing lines representing labels, but only for static text. Similarly, it is possible to add the label of a button via the annotations.

One possible solution for managing text would be to add another drawing mode, (for instance the “text mode”), which should recognize any gesture as text and convert it into label with the desired content. This functionnality would allow a complete export into UsiXML and seems more intuitive for the final user. Note however, that the widget "label" will then serve as dynamic text fields: “we knew that we would put one, but we did not know what it would contain in advance”.

We consider that the delete link in the navigation view is not intuitive: the final user need to redefine the link to get the "delete" menu of links is not practical. An alternative is to use the same mechanism as the rubber used in the drawing view.

The absence of borders on the drawing area also could be a troublemaker. Some users might prefer to have a clear view of the edges on a prototype, to have a more precise idea of the final UI.

Finally, we consider adding the ability to duplicate the last widget designed to place it several times on the windows faster. This functionality could complete the functionalities of copy and paste.

6. Benefits and Shortcomings

6.1 Flexible prototyping by sketching of Graphical User Interfaces

The popularity of touch-devices to enable users to insert information directly on the screen, using their fingers or a pen, instead of a mouse is increasing and they are getting more and more common. The paper and pencil approach is without doubt the fastest for flexible and rapid prototyping: no constraints are to be respected. However, its automated reuse is very difficult.

UsiSketch combines a pointing technology with pattern recognition techniques and combination of shapes to allow the recognition of drawn objects. This recognition allows dynamically changing the behavior of the prototyped interface, and exporting objects in a usable format in the later stages of software development.

6.2 A Cross-platform tool

UsiSketch is a Java GUI designer built as an Eclipse plug-in which supports multiple computing platforms. It incorporates a new mechanism that recognizes User Interface by modeling sketches on very large interaction surfaces. The tools provide an environment to allow users to insert objects in compliance with predefined grammars by capturing their gestures, typically using a pen on a tablet.

6.3 A standardized export format

In UsiSketch, any gesture representation is expressed in an XML format stored in a graphical grammar. UsiSketch transforms the users' representation into processable User Interface. The export process is based on the UsiXML language. The promised universality makes good support for this functionality.

6.4 Recognition of widgets and other gestures

UsiSketch has the ability to recognize shapes and combinations of shapes. Actually, UsiSketch recognizes and interprets 8 basic predefined shapes (i.e., triangle, rectangle, line, cross, wavy line, arrow, ellipse, and

circle); 32 different types of widgets (ranging from check boxes, listboxs, textfields, buttons, video multimedia, ...), and 6 basic commands (i.e., undo, redo, copy, paste, cut, new window).

Users of different domains can combine multiple simple shapes in a more complex combination or widget, according to pre-established rules. The recognition is done at the time of drawing, and not at the end thereof. The high fidelity elements are widgets. These are very numerous, and new ones appear frequently. Therefore, and since the recognition algorithms are based on a supervised learning, wanting to recognize these individual widgets requires a training phase of the tool for each of them. Such a process would be tedious, and will never be completed: for every new widget, we should repeat the process.

We have addressed the solution using the representation of a low fidelity widget as a composition of simple geometric shapes. For this reason, we decided to implement pattern recognition only on these geometric forms, greatly reducing the number of shapes to recognize. This reduces the time needed to train the algorithm. These forms are then combined in predefined grammars.

UsiSketch does not support mapping a text for a widget. This function must be carried out through text recognition. This kind of recognition uses other algorithms. We plan to consider this item in future software upgrades.

6.5 Multiple representations for a widget

UsiSketch has the possibility to recognize and interpret several representations for a widget at run-time. The contextual grammars used to define forms as a composition of simpler shapes allow conceiving a mechanism to define multiple representations of a widget. In this way, any custom object could be easily added by adding a new representation in the grammar. Theses representations can be structured in a hierarchical way. The hierarchy can then be augmented or modified with new representations.

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. The objects that are correctly recognized are beautified and the name is added. If an object is not recognized, it is simply maintained as it is, but could be annotated for further handling in the future.

6.6 Multiple levels of fidelity for a gesture

Figure 4 illustrates the fonctionality allowing the user to change the level of fidelity of the prototyped user interface. UsiSketch supports many widgets in four fidelity levels: none: (only the drawing is displayed), low fidelity (the drawing is displayed with recognized portions), medium fidelity (the drawing is beautified where portions are recognized, including the predefined shapes), and high fidelity (a genuine user interface is produced with widgets for those recognized objects).

6.7 Performance of recognition when the UI has many shapes

UsiSketch allows changing old widgets without having to redraw them. Our tool accommodates several representations for a single object, without affecting significantly the system response time. A combination of shapes depending on the drawing order does not allow this feature. However, the execution time of a research combination increases with the number of shapes drawn on a window. Although optimizations offered by constrained programming severely limit this effect, we believe that a window containing a large number of shapes can slow the performance of the combination. We expect to achieve a comprehensive testing to get results in such estimates and find affordable solutions.

6.8 User Interface dynamics

Currently the only proposed dynamic behavior is the click of a button. It is not possible to change the text of a field, check a box or other dynamic behavior. In addition, the prototyped user interfaces however remain very unrepresentative. We note that the operation of UsiSketch is limited for highly dynamic interfaces. For instance, when the user interface has incorporated a google maps widget. The only solution at present would be to copy each page and simulate the dynamic navigation. Although it is not vital to use our tool, improving this feature would be good.

6.9 Multiple fidelity transition

The User Interface of UsiSketch has a slider to give users the functionality to easily and quickly change between the four fidelity levels. A UI in the low fidelity mode is often referred to as a wireframe

representation, independent of any particular technology. A high fidelity mode displays genuine belonging to the Java platform. Different widget sets and look & feel could be used alternatively that mimic a high fidelity representation in other window managers and operating systems like Linux, Open Look, and MacOS X. If a UI element has not been recognized, it is simply kept as it is.

6.10 The use of multi-strokes

During the development of the tool, we found that the use of a multi-stroke features does not allow users to draw at the speed they want. That is to say, after some time without any stroke (about 300 ms), the software considers that the gesture is completed. An alternative that seems more effective is to use a minimum and maximum time window, after each stroke. The recognition is performed on it. If a stroke is drawn before a certain time, recognition returns the most similar shape. If, and only if, this second recognition returns a most similar form, the gesture is considered multi-strokes.

6.11 Additional features

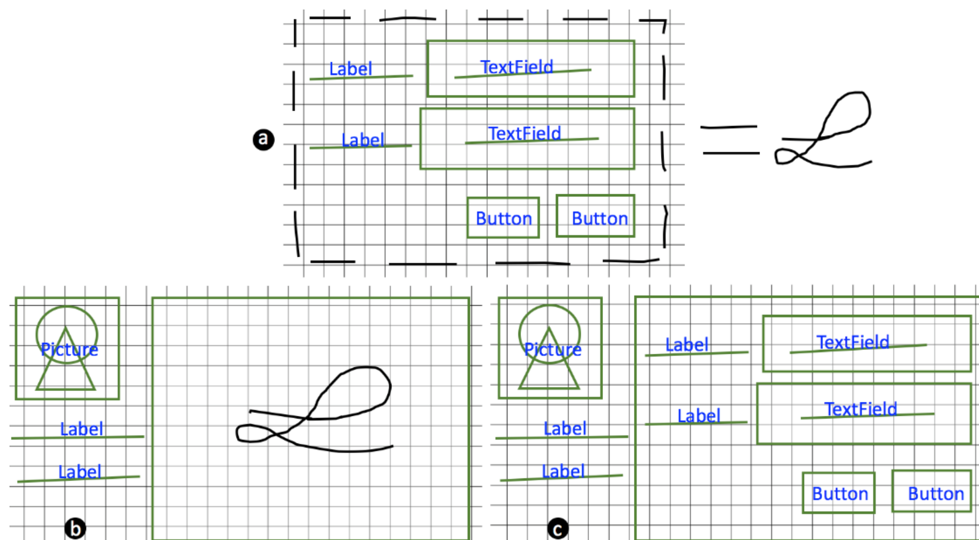

Figure 6. Illustration of the use of blocks. (a) Definition of block. (b) An example of user interface. The rectangle at the right is the bloc defined in (a). (c) The final UI with an intermediate fidelity level: the gesture is remplaced by block.

The major features are currently still lacking the ability to import an image and convert it into UsiSketch file. We also intend to develop an edition interface for generic grammars with visualization of their representative image. We want to achieve the functionality to duplicate the last widget designed, to place it several times faster.

We also intend to achieve a feature to define *blocks*. Figure 6 shows the representation of a block. Each block would have its specific content and would be identified by a gesture chosen by the user.

The user can then add a block on any window drawing the representative gesture of the block, and by framing the gesture by a rectangle to specify the block borders. A block can be seen as an element of type “box” in the syntax of UsiXML. It ensures the export to UsiXML. The scenario for creating a block would be:

1. The user informs the software that he wants to create a block.
2. The software asks the user to define the gesture representing the block. The gesture is then added as a training gesture to the pattern recognition module.
3. A new drawing area is created and the user can begin to describe the block content.
4. After setting the block, the user must choose the window(s) where the block can be placed and draw the gesture representing the block framed by a rectangle.

It may be necessary to insert an intermediate level of fidelity between “Low” and “Medium”, where the representative forms of a block would be replaced by their contents (see Figure 6).

7. Conclusion

We have presented an architecture for UsiSketch conceived from a list of 42 requirements. UsiSketch is a tool that supports horizontal prototyping. It provides collaborative design of user interfaces, even in very large surfaces where final designers conceive in a consensual manner the UI of a system.

UsiSketch takes advantage of different technologies, both software and hardware. Indeed, the software itself is a sketch prototyping tool which involves image recognition technology as well as prototyping tools. UsiSketch is easily accessible and does not require prior learning. It has the ability to represent many fidelity levels for a UI and also provide and

simulate the behavior of a prototype representing the final UI as closely as possible. The solution also relies on usiXML which is a description language for graphical user interfaces. It considers the capability of the tool to provide directly reusable resources that will be useful in later stages of the development process, avoiding in this way an additional cost.

As we have seen in the paper, the tool now allows doing what it was designed for: rapid prototyping with multiple fidelities. However, its full potential remains to be developed. The reflections during the development phase and informal tests are a first base of work to expand the software requirements and improve the software in parallel. We consider to extending the activity to sketching in another domains of human-computer interaction, specifically, we are interested in extending the UsiSketch tool to support the design of task models. Finally, we will be able to easily evaluate the feasibility of our tool by conducting user experiments. The results will be used to evaluate the performance of the tool, and obtain new research perspectives.

References

Ambler, S. W. (2007). Agile adoption rate survey results : March 2007. Retrieved from http://www.ambysoft.com/surveys/agileMarch2007.html

Caetano, A., Goulart, N., Fonseca, M., Jorge, J. (2002) JavaSketchIt: Issues in Sketching the Look of User Interfaces. In *Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding* (Palo Alto, March 2002). AAAI Press (2002) 9–14.

Cherubini, M., Venolia, G., DeLine, R., & Ko, A. J. (2007). Let's go to the whiteboard: How and why software developers use drawings. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* CHI'2007 (pp. 557–566). New York, NY, USA: ACM. doi:10.1145/1240624.1240714.

Coyette, A., Faulkner, S., Kolp, M., Limbourg, Q., & Vanderdonckt, J. (2004). Sketchixml: Towards a multi-agent design tool for sketching user interfaces based on usixml. In *Proceedings of the 3rd Annual Conference on Task Models and Diagrams* TAMODIA'2004 (pp. 75–82). New York, NY, USA: ACM. doi:10.1145/1045446.1045461.

Guerrero-Garcia, J., Gonzalez-Calleros, J. M., Vanderdonckt, J., & Munoz-Arteaga, J. (2009, November). A theoretical survey of user interface description languages: Preliminary results. In Web Congress, 2009. LA-WEB'09. Latin American (pp. 36-43). IEEE.

Hammond, T., & Davis, R. (2002). Tahuti: A geometrical sketch recognition system for uml class diagrams. In *AAAI Spring Symposium on Sketch Understanding* (pp. 59–68). Stanford, California: AAAI Press.

Kim, H., Kwak, K., Jung, J., Myung, I., & Hahn, M. (2010). Tablaction: Col- laborative brainstorming system with stylus-fingertip interactions on tablet pcs. In *Proceedings of the 9th ACM SIGGRAPH Conference on Virtual- Reality Continuum and Its Applications in Industry* VRCAI '2010 (pp. 81–84). New York, NY, USA: ACM. doi:10.1145/1900179.1900195.

Klemmer, S. R., Newman, M. W., Farrell, R., Bilezikjian, M., & Landay, J. A. (2001). The designers' outpost: A tangible interface for collabo- rative web site. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology* UIST'2001 (pp. 1–10). New York, NY, USA: ACM. doi:10.1145/502348.502350.

Landay, J.A., Myers, B.A. (1995). Interactive Sketching for the Early Stages of User Interface Design. In *Proceedings of ACM Conference on Human Factors in Computing Systems*, CHI'1995, Denver, 7-11 mai 1995, ACM Press, New York, 43-50.

Landay, J.A., Myers, B.A. (2001). Sketching Interfaces: Toward More Human Interface Design. *IEEE Computer*, vol. 34, num. 3, 56-64.

Lin, J., & Landay, J. A. (2002). Damask: A tool for early-stage design and prototyping of multi-device user interfaces. In *Proceedings of The 8th Inter- national Conference on Distributed Multimedia Systems* (2002 International Workshop on Visual Computing) (pp. 573–580).

Lin, J., & Landay, J. A. (2008). Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* CHI'2008 (pp. 1313–1322). New York, NY, USA: ACM. doi:10.1145/1357054.1357260.

van der Lugt, R. (2002). Functions of sketching in design idea generation meetings. In *Proceedings of the 4th Conference on Creativity & Cognition* C&C'2002 (pp. 72–79). New York, NY, USA: ACM. doi:10.1145/581710.581723.

Mitra, R. (2015). Rapido: A sketching tool for web api designers. In *Proceedings of the 24th International Conference on World Wide Web* WWW '15 Companion (pp. 1509–1514). Geneva, Switzerland: International World Wide Web Conferences Steering Committee. doi:10.1145/2740908.2743040.

Newman, M.W., Lin, J., Hong, J.I., Landay, J.A. (2003). DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice. Human-Computer Interaction, vol. 18, 259-324.

Nielsen, J. (1993). Prototyping. Chapter 4. In Usability Engineering, Nielsen, J. (Ed.), Academic Press, 93-101.

Norman, D. A., & Draper, S. W. (1986). *User Centered System Design; New Perspectives on Human-Computer Interaction*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc.

Obrenovic, v., & Martens, J.-B. (2011). Sketching interactive systems with sketchify. *ACM Trans. Comput.-Hum. Interact.*, 18 , 4:1–4:38. doi:10.1145/1959022.1959026.

Oehlberg, L., Simm, K., Jones, J., Agogino, A., & Hartmann, B. (2012). Showing is sharing: Building shared understanding in human-centered design teams with dazzle. In *Proceedings of the Designing Interactive Systems Conference* DIS'2012 (pp. 669–678). New York, NY, USA: ACM.doi:10.1145/ 2317956.23180 57.

Pérez Medina, J. L. (2016) Methods for Modelling Sketches in the Collaborative Prototyping of User Interfaces. *Revista Romana de Interactiune Om-Calculator* 9(3), 183-216.

Pérez Medina, J. L., & Vanderdonckt, J. (2016). A Tool for Multi-Surface Collaborative Sketching. Workshop on Cross-Surface at ISS2016. Niagara Falls, Canada, 06-09 november 2016.

Petrie, J.N., Schneider, K.A. (2006). Mixed-Fidelity Prototyping of User Interfaces. In *Proceedings of 13th International Workshop on Design, Specification, and Verification of Interactive Systems*, DSV-IS'2006, Dublin, 26-28 juillet 2006, Lecture Notes in Comput- er Science, Springer-Verlag, Berlin.

Safin, S., & Leclercq, P. (2009). User studies of a sketch-based collabora- tive distant design solution in industrial context. In Y. Luo (Ed.), *Cooperative Design, Visualization, and Engineering* (pp. 117–124). Springer Berlin Heidelberg volume 5738 of *Lecture Notes in Computer Science*. doi:10.1007/ 978-3-642-04265-2_16.

Sangiorgi, U., & Vanderdonckt, J. (2012). Gambit: Addressing multi-platform collaborative sketching with html5. In *Proceedings of the 4th ACM Symposium on Engineering Interactive Computing Systems* EICS'2012 (pp. 257–262). New York, NY, USA: ACM. doi:10.1145/2305484.2305527.

Sangiorgi, U. B., & Barbosa, S. (2010). Sketch: modeling using freehand drawing in eclipse graphical editors. *In FlexiTools 2010: ICSE 2010 Workshop on Flexible Modeling Tools*, Cape Town, South Africa.

Sangiorgi, U. B., Beuvens, F., & Vanderdonckt, J. (2012). User interface design by collaborative sketching. In *Proceedings of the Designing Interactive Systems Conference* DIS'2012 (pp. 378–387). New York, NY, USA: ACM.

Tuddenham, P., Davies, I., & Robinson, P. (2009). Websurface: An interface for co-located collaborative information gathering. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces* ITS'2009 (pp. 181–188). New York, NY, USA: ACM. doi:10.1145/1731903.1731938.

UsiXML Consortium. UsiXML, a General Purpose XML Compliant user Interface Description Language, *UsiXML V1.8*, 23 February 2007. Available on line: http://www.usixml.org.

Vanderdonckt, J., & Coyette, A. (2007). Modèles, méthodes et outils de support au prototypage multi-fidélité des interfaces graphiques. *Revue d'Interaction Homme-Machine*.

West, D., Seyff, N., & Glinz, M. (2013). Flexisketch: A mobile sketching tool for software modeling. In D. Uhler, K. Mehta, & J. Wong (Eds.), *Mobile Computing, Applications, and Services* (pp. 225–244). Springer Berlin Heidelberg volume 110 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. doi:10.1007/978-3-642-36632-1_13.

Wuest, D., Seyff, N., & Glinz, M. (2015a). Flexisketch team: Collaborative sketching and notation creation on the fly. In *IEEE/ACM 37th IEEE International Conference on Software Engineering* ICS'2015 (pp. 685–688). volume 2. doi:10.1109/ICSE.2015.223.

Wuest, D., Seyff, N., & Glinz, M. (2015b). Sketching and notation creation with flexisketch team: Evaluating a new means for collaborative requirements elicitation. In *Proceedings of IEEE 23rd International Conference on Requirements Engineering* RE'2015 (pp. 186–195). doi:10.1109/RE.2015.7320421.

Appendix

1. Recognition requirements

1	During the design of the UI, UsiSketch must be able to recognize a geometric shape and convert it into a smoothed form. The forms to be recognized are: line, arrow, wavy line, triangle, rectangle, circle, cross). A smoothed form has a higher fidelity level that its equivalent freehand-drawn. It is also necessary for the combination of forms.
2	During the design of the UI, when a new form is recognized the software must be able to limit all forms of recognition, that is, to minimize the learning process for recognition. This means that UsiSketch must be able to combine multiple simple forms in a more complex form or widget, according to pre-established rules. Every time a new form is added to a window, the software must check whether it is possible to combine it with other forms.
3	During the training recognition phase, the developed recognition algorithm must learn from mistakes and adapt to its user. If there is an error in the recognition, the designer must be capable to report the error.
4	During the design of the UI, when a new form is recognized and several combinations are possible but are mutually exclusives. UsiSketch must be capable to decide what is the combination of forms to choose, and have the same behavior at any time. The most constrained combination is by definition more difficult to validate, it must be the priority. If two mutually exlusive combinations are possible based on a list of forms, the most constrained combination should be selected.
5	When the designer needs to define a new widget or change a composition rule, the software must have the ability to perform it outside the software code. This allows great flexibility of the tool in the definition of compositions, allow the designer to only follow the composition rules that suit him.
6	The designers must edit composition rules using a simple and intuitive interface without programming.
7	If the designer already has a paper prototype and wants to use it, UsiSketch must be able to import low fidelity paper prototype or an image drawn in a UsiSketch project.
8	During the design of the UI, the designer will have the ability to edit the text description of widgets rather than have to do it later in an editor at the hightest fidelity level. For that, UsiSketch must be able to recognize the manuscript text and convert it into its computer equivalent.

2. Drawing – Rendering requirements

9	UsiSketh must always be consistent with the current model drawing. UsiSketch must be compliant with “what you see is what you get” (WYSIWYG).
10	Some widgets could be unique and specific to a UI. In this case, it is impossible to represent all situations. However, the tools must therefore leave more free space for the designers. UsiSketch must have a design mode based on recognition. After each drawing, the application must analyze it and return the recognized vector shape. The designer may also want to give a visual feedback, via annotations.
11	Concerning the above requirement; UsiSketch must also incorporate an annotation mode where no recognition is made.
12	The desired user experience must be as natural as possible. Many people can draw certain shapes with several strokes (for example: when a designer sketches a cross). This requires the recognition algorithm to be able to handle it. As a consequence, UsiSketch must support multi-trait recognition.
13	The designer must be able to delete a gesture placed by mistake or when it become obsolete.
14	UsiSketch must offer a way to select a widget or group of widgets and allow the designers to move, copy or delete them.
15	UsiSketch should allow moving a selection when designers require aligning widgets, or moving them to a different area of the windows without having to erase and redraw them.
16	Concerning the above requirement; UsiSketch must offer a way to copy, cut and paste a selection.
17	A lower fidelity level is more pleasant during the drawing. Conversely, a high fidelity level is more pleasant during the creation of interactions between windows and widgets. UsiSketch must allow incorporating multiple fidelity levels, changeable at any time by the user. The fidelity levels must be: (1) drawing fidelity: the designs are displayed as were drawn, nothing else; (2) form fidelity: recognized forms are displayed smoothed, widgets are not displayed; (3) fidelity widgets: widgets are displayed as representative images, the remaining forms as smooth representations; (4) fidelity simulation: widgets are displayed as real widgets, the remaining forms in smoothed representation.

3. Navigation requirements

18	During the Interface design, typical actions on a widget or window (e.g. show, hide, minimize, maximize, setText, …) must be defined. It allows more faithful simulability than just a display of statically windows.
19	The design of interactions between widgets and the simulation requires that typical events/actions for a widget (like as onClick or onChange events) must be set. This will perform certain actions of a widget.
20	The designer can increase the fidelity of the prototype without having to use a keyboard.
21	UsiSketch must be able to structure the graphic elements in the form of a tree. It must allow desingers to work on a structured prototype, where the containers contain their widgets, rather than a “flat” view, where containers have only a visual function.

	In other words, the widgets must be added to a container of type widget, automatically or manually. The structure should be compatible with the project UsiXML.

4. Data requirements

22	The interface is designed to be exported in a standardized format, allowing editing the result with a higher fidelity editor, chosen by the designer. The software must be able to export a usiSketch project into a UsiXML file.
23	UsiSketch must be able to export a project into another representation than UsiXML. This requirement aims to ensure high system interoperability.
24	UsiSketch must be able to import a UsiXML file and convert it into UsiSketch format when part of the prototype has been defined through another compatible UsiXML software, or when the source file in UsiSketch format is not available. It may be convenient to recycle an old UI and change it, or just annotate it. It avoids to re-design a complex interface if a similar interface has already been defined.

5. Simulattion requirements

25	During the interface design, an interface defined in UsiSketch must be simulated summarily. It means that if the designer wants to test its interface, UsiSketch must allow the designer to test/simulate the interface without having to export it into UsiXML file every time.
26	When a designer wants a review of its current design prototype. A UI defined in UsiSketch must be sent from a designer to testers for simulation and feedback.
27	Each tester must be able to give one or more feedback when testing a prototype.
28	The feedbacks provided are used to improve the prototype. This requires designers to have access to a list of feedbacks provided by the(s) tester(s).

6. Ergonomy requirements

29	UsiSketch must be simple to learn and use. The learning curve of using the software be as fast as possible. The designers should at ease from the first contact with the software.
30	UsiSketch should be designed for use with tablets. It is necessary to limit the exchanges stylus-keyboards or stylus-mouse. Conversely, as many prototyping operations as possible must be accessible through the pointer.
31	Buttons smaller than 50x50 pixels are unpleasant to use with a stylus. The buttons on toolbars must be sufficiently large to be easy to click. A 50x50 pixel button size is an acceptable size.
32	In order to see the evolution of a designed UI, and also to allow backtracking, UsiSketch must maintain a desing of history and drawing of a prototype at any time.
33	The designer must be able to undo/redo an action contained in the design history of the prototype.
34	Historic of actions must be kept and backward steps should be allowed any time even after close. Furthermore, this ensures a good reconstruction of the UI. The log

	history can be compressed to reduce the size of the prototype and reduce time spent for reconstruction. A prototype saved under UsiSketch format shall a compressed design and drawing history.

7. Architecture requirements

35	UsiSketch must be developed as an Eclipse plugin. Eclipse environment is a well-known tool for developers. This should facilitate the use and learning of the tool.
36	UsiSketch can run on any computer device and should be useable regardless of a specific operating system.
37	The tester has no right to modify the interface. He does not need to access the drawing functionality, but only to access the navigation mode.
38	The drawing/design and simulation modules should be developed as two separate programs.
39	UsiSketch does not integrate code or library under GPL or LGPL. These licences are highly copyleft, which means that their integration requests the software to be distributed under the same license.
40	If a new widget appears and is widely used, UsiSketch must allow to incorporate the use of new widgets. The architecture must be designed to facilitate the addition of new types of widgets.
41	Research in shape recognition remains very active and a more efficient method may thus be found in the future. UsiSketch must guarantee a possible migration to the new method at lowest cost. An easily replaceable module for UsiSketch also allows designers to test different algorithms more easily. The recognition module should be easily replaceable by a module that implements another algorithm.
42	UsiSketch should consider that final users play the role of testers during the UI prototyping. The test module should be available online, and ideally without any need of installation, so, typically a Web interface.