Revista Română de Interacțiune Om-Calculator 7 (2) 2014, 139-160

UsiGesture: Test and Evaluation of an Environment for Integrating Gestures in User Interfaces

François Beuvens, Jean Vanderdonckt

Université catholique de Louvain, Louvain School of Management - LILab
1 Place des Doyens, 1348 Louvain-la-Neuve, Belgium
E-mail: francois.beuvens@uclouvain.be, jean.vanderdonckt@uclouvain.be

Abstract. User interfaces allowing gesture recognition and manipulation are becoming more and more popular these last years. It however remains a hard task for programmers to developer such interfaces : some knowledge of recognition systems is required, along with user experience and user interface management knowledge. It is often difficult for only one developer to handle all this knowledge by itself and it is why a team gathering different skills is most of the time needed. We previously presented a method along with a tool in order to ease the collaboration between members of such a team. In this paper, we present results and feedbacks collected by observing different teams that followed the method and/or used the proposed tool.

Keywords: HCI, recognition, pen-based interaction, User interface modeling.

1. Introduction

Gesture-based user interfaces are getting more popular last years with the emergence smartphones, tablets, and any other flat interaction surface that could accommodate pen-based gestures. These new platforms usually require gesture-based interaction with – often but not always – finger or pen as inputs. Despite their recent increased popularity, such user interfaces are considered for a long time and several tools have been realized in order to bring support during their creation.

Pen-based gesture recognition (Calvary et al., 2003) (Landay, 1996) (Signer et al., 2007) typically consists in interpreting hand-made marks, called *strokes* (Beuvens and Vanderdonckt, 2012) (Long, 2001), made with a pointing device (e.g., a mouse, a stylus, a light pen) on a flat constrained vertical or horizontal surface (e.g., a table, a wall or a graphic tablet). Pen-based gestures are applicable to a large area of tasks (e.g., music editing,

drawing, sketching, spreadsheets, web navigation, equation editing) in many different domains of activity (e.g., office automation (Wolf, 1986), ambient intelligence (Hinckley et al., 2004), multimodal systems (Wilhelm et al., 2010)) and a growing set of devices, ranging from smartphones to tabletop interaction. Pen-based gestures can even be considered across several platforms: starting on a smartphone and finishing on a tabletop (Hinckley et al., 2004). When the locus of input is different from the locus of output (e.g., with a graphic tablet), gestures are drawn outside the main display, thus posing a visual discontinuity problem. When locus of input and output match, a risk of occlusion occurs since the gesture is drawn on top of the main display. The surface used for pen-based gestures is however used as a way to constrain the gesture, thus helping its recognition.

Pen-based gestures have received considerable attention in both research and development, namely for addressing the scientific problem of modeling, analyzing, learning, interpreting, and recognizing gestures in a large spectrum of setups. The large inclusion of pen-based gestures in widely-available interactive applications has however not reached its full potential due to at least the following reasons: designers and developers do not know which recognition algorithm to select from such as large offer, how to tune the selected algorithm depending on their context of use, and how to incorporate the selected algorithm into streamlined User Interface (UI) development in an effective and efficient way. Incorporating pen-based gestures may also involve using Application Programming Interfaces (APIs), libraries, toolkits or algorithm code that could be considered hard to use.

In our paper (Beuvens and Vanderdonckt, 2012) we propose a method and tool fostering team collaboration willing to create user interfaces including gesture manipulation. The proposed environment is not aimed at benchmarking algorithms like other platform like iGesture (Signer et al., 2007), Inkkit (Plimmer and Freeman, 2007) or DataManater (Schmieder et al., 2009). UsiGesture is a helper for the integration of gestures but is independent of the choice of gestures and targets. It targets an improved and streamlined development of gestural interface and is complementary with the benchmarking platforms.

The method presented for UsiGesture presented in Figure 1 defines 4 different types of stakeholders involved in the creation of such an interface:

- The **engineer/architect** analyzes the different requirements elicited by the user, the environment, or any other input and identifies the different parts to be included in the user interface (widgets) as well as the behavior enabling interaction with the user and between the widgets. He is taking care of the ergonomics of the system.
- The **designer** is in charge of the aesthetics of the user interface. His role is to choose the right layout parameters (size, color, font, etc.) for each part of the user interface, and follow aesthetics rules based on metrics such as density (Vanderdonckt, 2003) or balance (Ngo and Byrne, 2001). He helps the engineer/architect ensuring good ergonomics.
- The **gesture specialist** is devoted to the recognition mechanism specification with its different parameters.
- The **programmer(s)** are the builders of the user interface. Based on the specifications of the conception phase, they actually code it. This includes the recognition mechanism, i.e. the algorithms and the gesture datasets.

These roles are well delimited but they are usually more interfering and lead to a real cooperation. In practice they are not exclusive as a single person can play two or more roles. For example, the gesture specialist is not always available and present in the organization developing the interface and the developer of the interface is rarely well trained to this problem.

To support the method, a tool (see Figure 2) is proposed in order to formalize the collaboration between these stakeholders. For that purpose, 3 roles are defined regarding the tool and the user interface to be produced:

- The **Interface Users (IU)**: end users of the interface.
- The **System Users (SU)**: first group of engineer(s)/architect(s), designer(s) and gesture specialist(s) using the system in order to produce the user interface for the Interface Users.
- The **System Feeders (SF)**: second group of engineer(s)/architect(s), designer(s), gesture specialist(s) and programmer(s) feeding the system with knowledge allowing SU creating user interfaces.

The method is defined in 7 steps:

1. Interface Users define user interface requirements.

2. Based on the UI requirements, System Users define system requirements.
3. If system requirements not met: based on the system requirements, System Feeders feed the system.
4. Based on UI requirements, System Users use the system to produce the user interface.
5. If UI requirements not met: System Users refine system requirements, then go back to step 3.
6. Interface Users use the produced user interface.
7. If Interface Users not satisfied: Interface Users refine UI requirements, then go back to step 4.

To illustrate this method with a short example, we can consider the creation of a document reader manipulated by gestures. It has been ordered by John and Tom who are two big readers and want to improve their reading process. As the interface users, they elicit their requirements: the reader must have the ability to change the current page with gestures. Bob, the developer in charge of this project, is not aware of gesture recognition and wants to use UsiGesture tool that can help for that: he will act as a system user. Unfortunately, the current state of UsiGesture does not include such a feature. Bob will then contact Alice, a gesture specialist, to help him in his task. She will then act as system feeder by implement the “next” and “previous” gestures in UsiGesture tool. Bob will then be able to develop the interface, and John and Tom to enjoy an improved document reader.

This small scenario shortly shows how the three categories of actors can interact through UsiGesture. In practice, workflow is not so straightforward and the steps are redone multiple times.

UsiGesture tool is articulated around three main regions. The upper right part is the graphical representation of the interface being built by the system users, along with the controls allowing adding the widgets needed (graphical widgets, gestural components, …). This part allows the System Users to build the interface through a WYSIWYG (What You See Is What You Get) process. The bottom part is the XML representation of the interface and can be used to visualize the current generated code or to add directly components by writing code. The last part is the left column showing the different Java classes representing the engine behind the editor

used by the System Users. The System Feeders can modify these Java classes to enrich the interface generator.

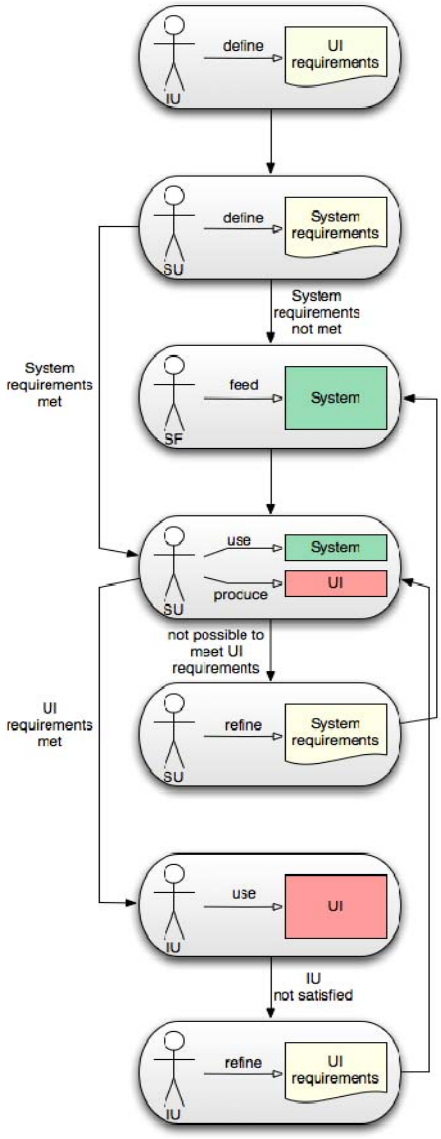

Figure 1 - UsiGesture method.

Figure 3 depicts the overview of the models used to describe the components currently available in the platform.

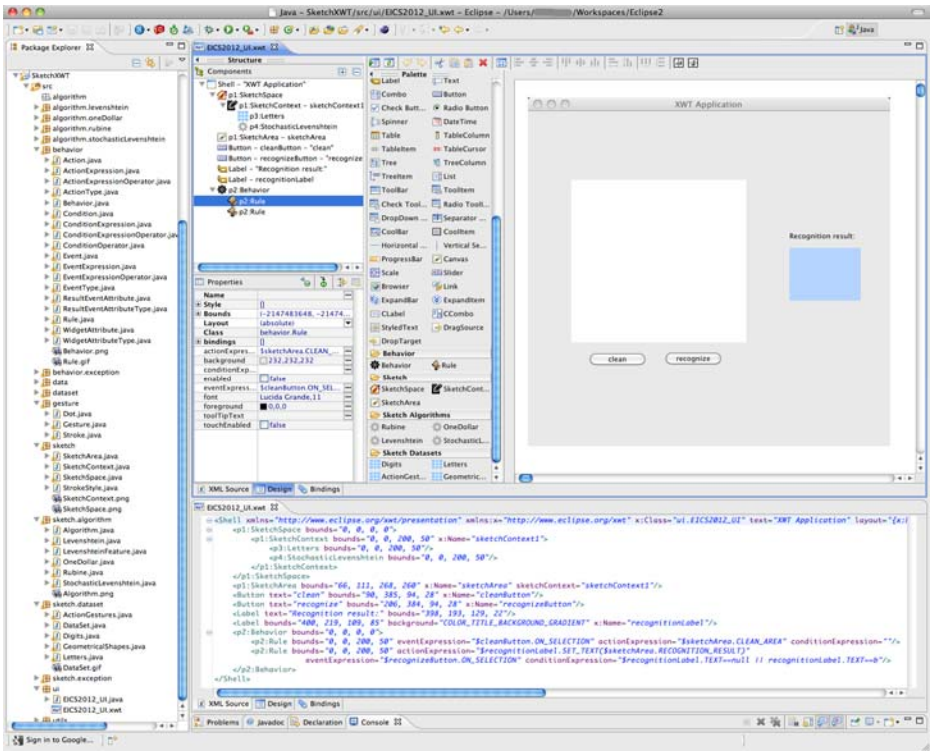

Figure 2 - UsiGesture tool.

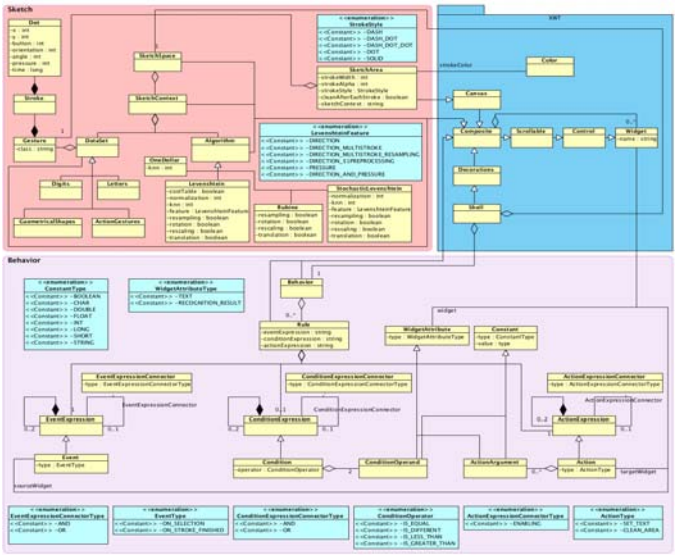

Figure 3 - UsiGesture model overview.

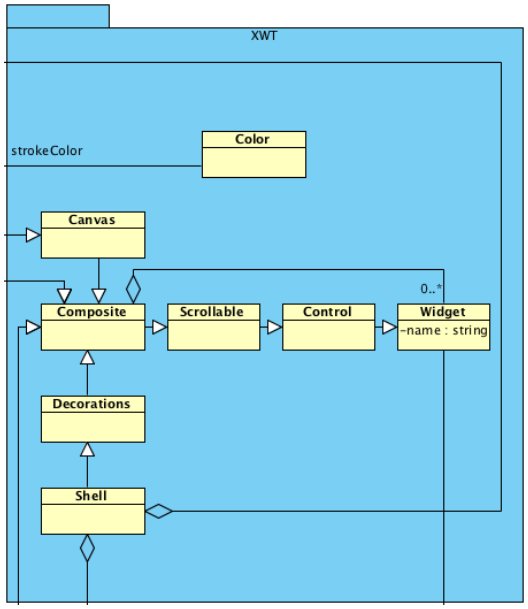

Figure 4 - Model core components.

The upper-right part represents the core functionalities package. The upper-left part (Figure 5) represents the gestural components, and the bottom part (Figure 6) the behavior ones. For more details, please refer to the aforementioned paper.

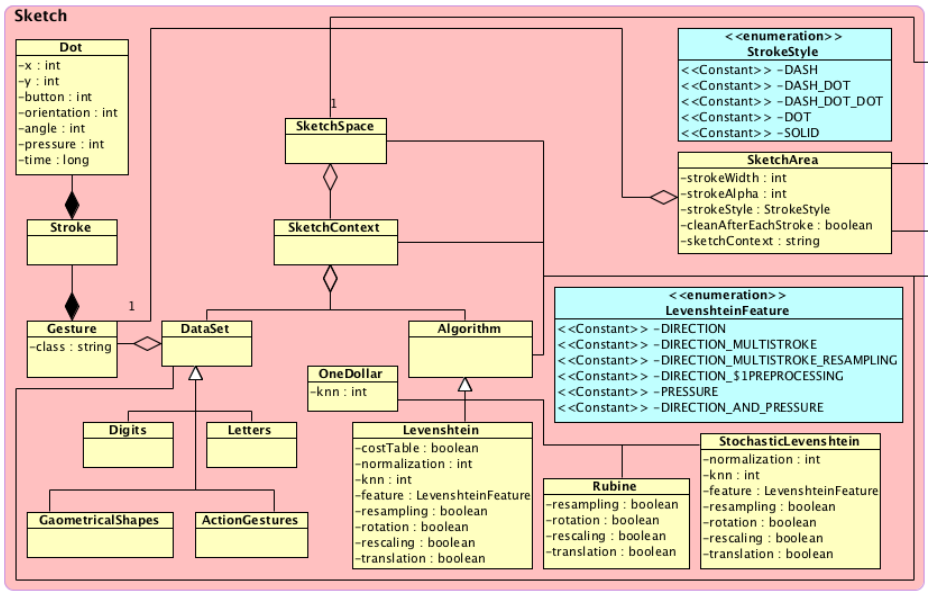

Figure 5 - Model gesture components.

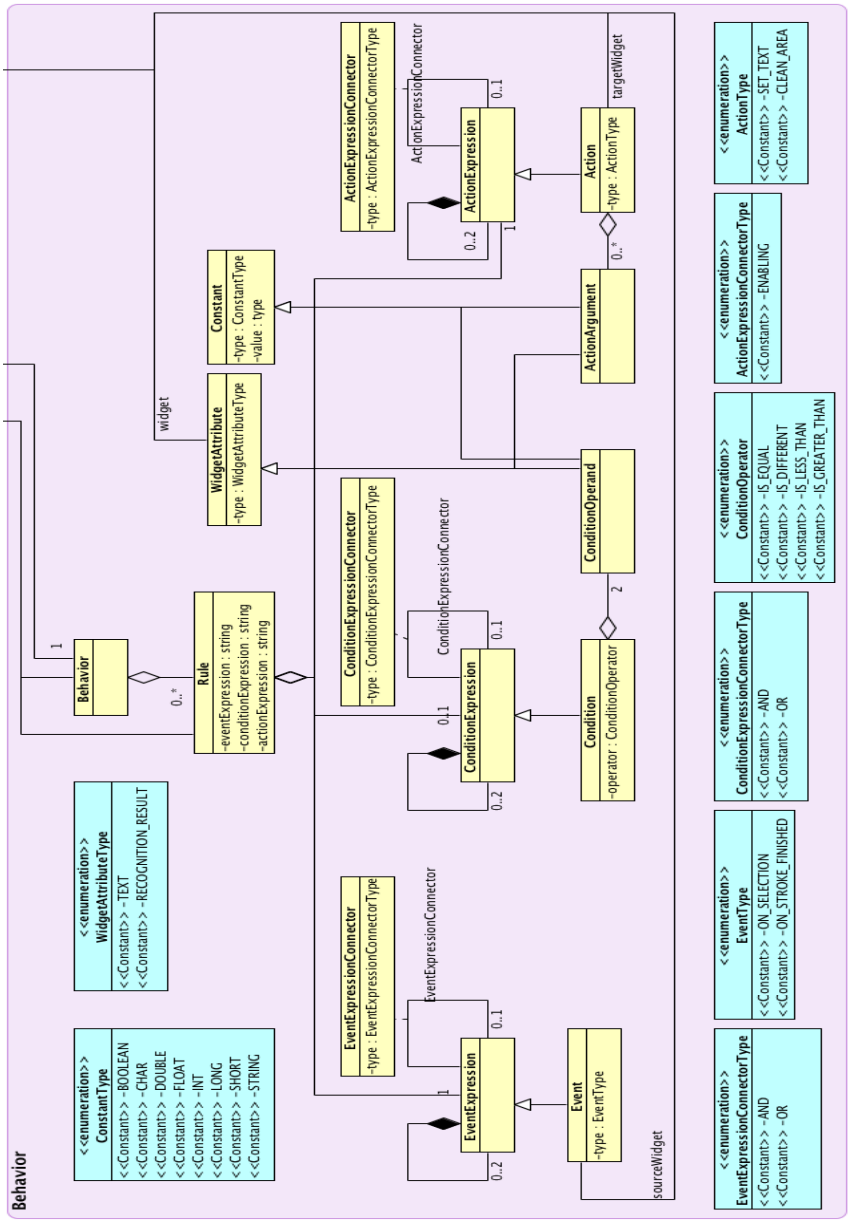

Figure 6 - Model behavior components.

2. Experiments

Two different experiments were conducted with the help of the method and the tool.

2.1 Woke

Description

The goal of this work was to create a user interface simulating the system of a woke restaurant (www.woke.be). It allows users to order food by following a step-by-step vegetable and meet picker. This system was created in two different ways:

- A more usual way where the actions are mapped to buttons, like minus and plus to specify the amount of items (see Figure 7).
- A gestural system where actions can be executed with gestures, like directly writing the amount of wanted items on the item itself.

The aim of these two representations is to compare usual interfaces with their gestural equivalent and think how to improve them by integrating gestures in a clever way. The scope of this work is limited to a portion of the whole woke system and covers the main task of ordering a menu, implying the three following subtasks:

- Choose ingredients: this task needs the user to define exactly what he would like in his menu. The interface can only say if it's possible or not (for instance, a user can't order 8 different meats at the same time).
- Validate the menu: the interface will display a kind of summary about what the user orders. It asks him to validate the menu if there is no problem.
- Prepare the menu and deliver bill: the interface's job here is only to display that the menu has been ordered.

Solution

The interfaces constituting the solution can be divided into 6 parts:

- A list displaying the summary of all ingredients chosen by the user, with the amount of each and the whole price.
- Another list displaying all available ingredients with the category.

- A label summarizing the number of calories, the items selected and the price of the menu.
- A button array allowing the user to browse in the categories and to confirm selection (confirmation window).
- A confirmation window.
- For the guidance, the interface shows a number indicating to the user the current step. That number is at the top of the product panel.

The main difference between classical and gestural interfaces is the way the number of items can be selected. With the classical ones, two buttons (labeled with "+" and "-") are used to increase/decrease the quantity. With the gestural ones, the quantity can be drawn on the pictures. For both interfaces, the number of each available ingredient is limited to 5.

The solution was created by respecting as much as possible the 6 criterions of Bastien and Scapin (Bastien and Scapin, 1993):

1. Coherence: the solution was thought to make it as easy as possible for the users, with recognizable actions.
2. Utility: the number of functionalities available. It is maybe a drawback of these interfaces, only a subset of them were implemented in the scope of this work;
3. Workload: the number of actions by task isn't too high;
4. Adaptation: this kind of interface (gestural) can be re-used in other contexts. This criterion is only used for gesture interface;
5. Representativity: each possible action has a label associated to it. Those labels are light and don't increase the workload;
6. Guidance: the solution was thought to avoid losing users in what they have to do.

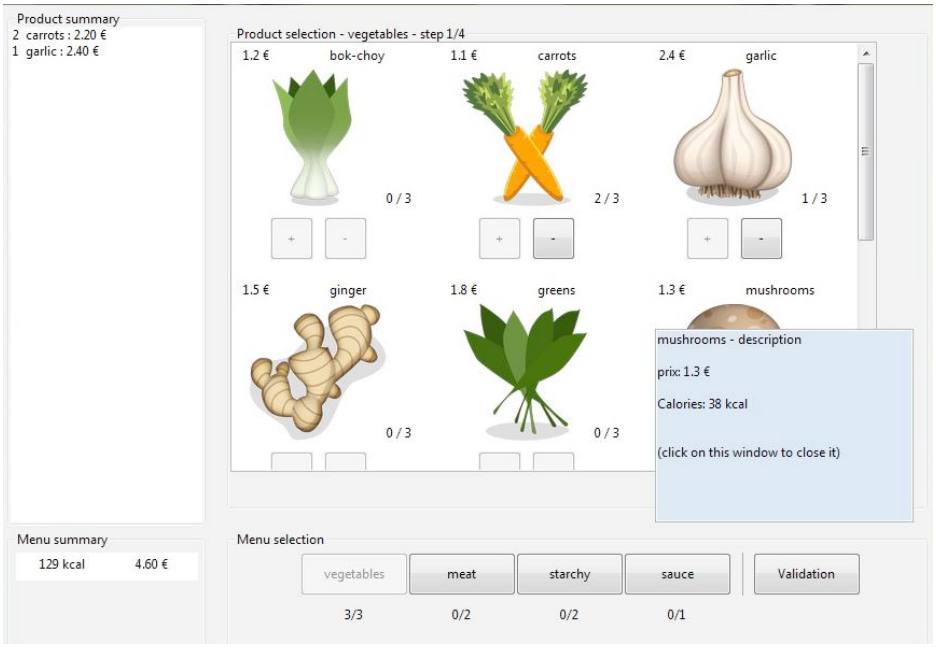

Figure 7 - Woke experiment (general interface).

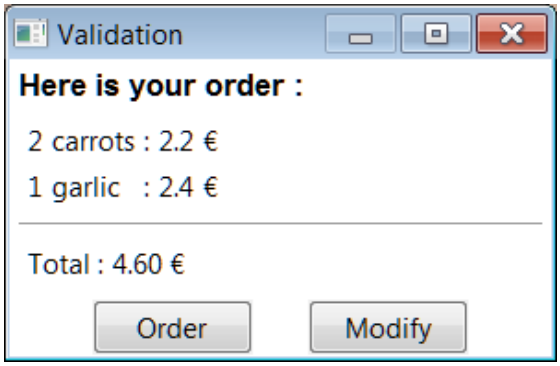

Figure 8 - Woke experiment (order interface)

In addition to user interface considerations, some tests have been conducted to select the best recognition engine. In order to reuse the mechanisms already provided by the platform, the algorithm to be used had to be selected among Rubine, Stochastic Levenshtein and One Dollar. For the woke use case, the most important gestures to be used are the digits and some action gestures. The tests have then been conducted on the 10 digits (0

to 9) and four geometrical shapes (square, triangle, circle, and rectangle). It appears the One Dollar algorithm was the most accurate for this case.

Survey and discussions

A survey was realized on 15 users for the comparison of classical and gestural interfaces (please refer to Appendix A for more information about these users). For this survey, participants were asked to rate the 6 ergonomics criterions of Bastien and Scapin explained before: coherence, utility, workload, adaptation (only asked for gestural interface), representativity and guidance. The results are depicted in Figure 9 and Figure 10.

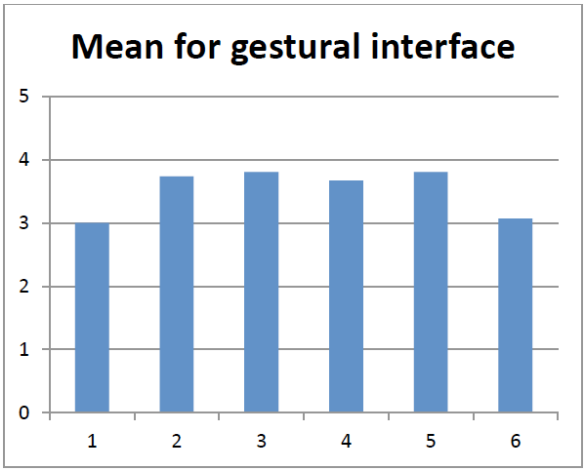

Figure 9 - Results for classical interface.

In addition to the graphics, general grades for each participant were computed from 0 to 5. By considering three levels "bad" (0 – 2), "average" (3) and "good" (4-5), we obtain:

- Classical interfaces: 9 good, 5 average, 1 bad;
- Gestural interfaces: 8 good, 6 average, 2 bad.

The results are quite comparable with a slight preference for the classical interfaces. The general feeling about the gestural interfaces was that they are more straightforward and intuitive, but the lack of recognition mechanism for some digits was decreasing the level of usability.

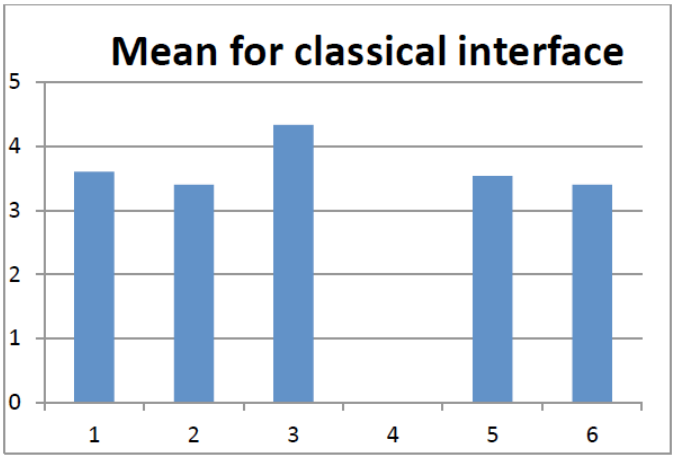

Figure 10 - Results for gestural interface

2.2. Restaurant

Description

The objective of this project is to assess two algorithms, Rubine (Rubine, 1991) and One Dollar (Wobbrock et al., 2007), in the situation of a restaurant application and more precisely through the different gestures used to navigate through this application.

The targeted task is similar to the woke interfaces. The goal is to select items to order, by specifying the quantity with numbers directly drawn on them.

Solution

The application is divided in two main sections: The drinks and the burgers. Each page of the application shows a set of choices to the client and these pages are divided into three parts as shown in Figure 11 and Figure 12.

Figure 11 - Restaurant experiment interface (drinks).

Figure 12 - Restaurant experiment interface (food).

The first part of the application is a toggle menu and is positioned on the top of the page. This menu shows the next and the previous page of the application, both the drinks and food menu, and can be triggered by the two gestures, left and right arrows.

The second part contains the user's choice of hamburgers, self-made hamburgers, beers, soft drinks, hot drinks and is the biggest part of the application. It allows the user or the waiter to write a number on each choice, this will then be registered in the third part of the application.

The third and last part is the client order list and is positioned on the right side of the second part. It allows the client or the waiter to clean or validate the order list with the two gestures, ok and no. These two gestures have been added to the original set of gestures and are represented in Figure 13 with the other numerical and arrow gestures used.

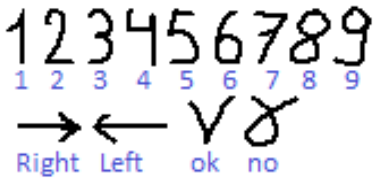

Figure 13 - Gesture set for restaurant experiment.

The application was made so that the user can quickly change the number of drinks he ordered. The value of each menu or drink is overwritten when the user draws a new digit on it. This method provides more usability to the user because a mistake can quickly be changed but it limits the maximum amount of each menu and drink to nine. This is because we can only recognize digits from zero to nine.

As an experiment, two same interfaces with different recognition engines (Rubine and One Dollar) were firstly created. Several people were asked to play with these two applications so that results could be collected separately for each algorithm. With these results, Rubine and One Dollar algorithms could be analyzed on every gesture and at the same time an improved version mixing both algorithms could be created.

During the experiment, people could use the mouse, a touchpad or an electronic pen with a tablet to test the application. Each person was asked to do the experiment with every restaurant application made to evaluate the two different recognition algorithms. The experiments was conducted either on a UNIX or Windows operating system, and can be visualized in Figure

14 where all numbers were drawn by an anonymous tester to test the effectiveness of gesture recognition.

Figure 14 - Restaurant experiment gestures handling.

Data has been collected on a total of 19 persons. The results presented in Table 4 show that One Dollar algorithm is better for the digits, the ok and the no gesture. The Rubine algorithm seems to be perfect for the arrows. We can see that the digits too have some troubles but it actually really depends on who draws the digits. For eight persons the results for the digit "2" were equal to 100%. Another interesting notice is that the result of the recognition was often the digit three when the result was wrong. These discoveries allowed understanding that by adding records closer to the writing of the eleven other persons to the recognition environment, better results can be obtained. For the Ok and No gestures, the Rubine algorithm works strangely as it recognize well the Ok when the gesture is small and the No when the gesture is big.

Table 4 – Restaurant experiment results

	Rubine	One Dollar
Digits		
1 (one)	72%	94%
2 (two)	77%	55,5%
3 (three)	100%	98%

4 (four)	0%	98%
5 (five)	0%	93%
6 (six)	11%	98%
7 (seven)	0%	80%
8 (eight)	100%	100%
9 (nine)	0%	98%
10 (ten)	72%	100%
Actions		
Right arrow	100%	2%
Left arrow	100%	2%
Ok	50%	100%
No	50%	100%

Survey and discussion

Along with the experiment, participants were also asked their opinion about the graphical user interface and its easiness.

From the graphical user interface standpoint, the nineteen users gave a lot of different feedbacks. Fourteen people found that the use of gestures for this application was really fun; three people found that it was interesting but they prefer the WIMP paradigm (Taylor, 2009). The last two people are completely opposed to the use of gestures such as digits because they think it is neither intuitive nor accurate. Almost every user has the same issue: they are not used to draw with a mouse and are not accustomed to work with a tablet. We think that this issue is more important than it seems because it is the habit of WIMP that will curb some people to use our application. Another point of the graphical user interface is that the use of the gesture “ok” and “no” is not so intuitive. If people do not know these two gestures they cannot clean or order something. In consequence we think that the application is more suitable for a waiter or a waitress than a simple client because they will have a learning curve for the two gestures to use.

It was also noticed that 75% of the people prefer to use their fingers or a digital pen than the mouse when using our application, and that it was more intuitive for most people to draw a line going to the left in the upper menu in order to go to the right side of the menu. It may be explained by the young age of participants and their ability to use their smartphone.

3. Discussion and conclusion

This paper reports two experiments of gestural exploration for graphical interfaces. They allow highlighting some needs for gestural interfaces confection and evaluation. The main advantage brought by UsiGesture platform in these situations is the possibility to avoid spending time on recognition mechanisms and only focus on the most important: ergonomics of the interface. UsiGesture acts as repository of algorithms as well as data samples allowing setting the recognition system, but it does not assess the right gestures for the right actions.

One important criterion to take into account when designing such interfaces is the compatibility between the real world and the system. Gestures need to be chosen adequately in order to be as representative as possible of what human being would expect. This point has a big impact on coherence criterion from Bastien and Scapin.

Different criterions of SQuaRE (Software Product Quality Requirements and Evaluation) (ISO/IEC 25000) are or can be impacted by UsiGesture method. Here are the most important ones:

- Interoperability: Interoperability is the ability of making systems and organizations work together (inter-operate). UsiGesture improves collaboration between different kinds of stakeholder (gesture specialists, developers, designers, end-users, …).
- Suitability: Suitability is the degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions. UsiGesture tool is targeted to help to the integration of gestures and improves this process if used according the proposed method.
- Resource utilization: Resource utilization is the ability to improve available resources. UsiGesture improves human resources utilization by allowing a good separation of tasks assigned to the suitable stakeholders.
- Changeability: Changeability is the degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality. Adding new gestures algorithms or dataset in UsiGesture is quite straightforward. For adding gesture, the feeders needs to create the

recognition mechanism itself and then plug it to the platform by creating one new class and modifying two methods. The plugin would require an effort estimated to of a few hours. Extending the datasets represents a similar effort. For the user point of view, changing an algorithm or dataset on a previously created interface is very simple it can be changed through an option in the WYSIWYG editor.

UsiGesture method can bring a good structure for the development lifecycle of a project, but it may not be necessary in small applications. The developers for the woke use case made the decision to not use it and develop the interface independently of the end users with a result not very convincing. However, the developers of the restaurant use case tried to take end users into account by selecting the right algorithms for the right gestures and then improving user experience.

Selecting the best algorithms to improve recognition is an example of many possibilities to improve the user experience, like selecting the appropriate gestures for the right situation, increasing the gesture set or the number of existing algorithms. The platform is already designed to easily support the improvement of algorithms and gestures set knowledge by capitalizing on the different application coded with it. Additionally to the developers' knowledge, UsiGesture can still be improved by adding the ability to automatically capitalize on end users knowledge. For example, the platform could compute success rates between algorithms and types of gestures (and possibly by taking context into account) thanks to all previous recognitions done through the system. It may then support developers in their choices of algorithms and gestures by suggesting the best combinations. Another possibility would be to propose a survey for querying the user at the end of interfaces manipulations. This could be useful to aggregate quality information about Bastien and Scapin criterions and guide the developers during the creation process.

References

Bastien, J.M.C., Scapin, D. (1993) Ergonomic Criteria for the Evaluation of Human-Computer Interfaces. *Institut National de recherche en informatique et en automatique,* France (http://www.inria.fr)

Beuvens, F. and Vanderdonckt, J. (2012) Designing Graphical User Interfaces Integrating Gestures. *Proc. of SigDoc'12*, 313-322

Beuvens, F. and Vanderdonckt, J. (2012) UsiGesture: an Environment for Integrating Pen-Based Interaction in User Interfaces. *Proc. of RCIS'12*, 339-350

Calvary, G. et al. (2003) A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computer 15 (3)*, 289-308.

Hinckley, K., Ramos, G., Guimbretière, F., Baudish, P., and Smith, M. Stitching (2004) pen gestures that span multiple displays. *Proc. of AVI'04*. ACM Press, NY, 23-31

ISO/IEC 25000, http://www.iso.org/iso/catalogue_detail?csnumber=35683

Landay, J. A. (1996) SILK: Sketching Interfaces Like Krazy. *CHI'96, ACM, NY*, 389-399.

Levenshtein, V. I. (1966) Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady 10 (8)*, 707–710

Long, A.C.J. (2001) quill: A Gesture Design Tool for Pen-based User Interfaces. *PhD Thesis, Univ. of California at Berkeley*

Ngo, D. C. L., Byrne, J. G. (2001) Another Look at a Model for Evaluating Interface Aesthetics. *AMCS 11 (2)*, 515-535

Plimmer, B., Freeman, I. (2007) A Toolkit Approach to Sketched Diagram Recognition. *HCI, Lancaster, UK*, 205-213

Rubine, D. (1991) Specifying gestures by example. *SIGGRAPH Computer Graphics*

Schmieder, P., Plimmer, B., and Blagojevic, R. (2009) Automatic Evaluation of Sketch Recognizers. *SBIM'09 ACM Press*, 85–92.

Signer, B., Kurmann, U., and Norrie, M. C. (2007) iGesture: A General Gesture Recognition Framework. *ICDAR'2007, Los Alamitos*, 954-958.

Taylor A. G. (2009) WIMP interfaces, http://www.cc.gatech.edu/classes/cs6751_97_winter/Topics/dialog-wimp/

Vanderdonckt, J. (2003) Visual Design Methods In Interactive Applications. *Mahwah : Lawrence Erlbaum Associates,* 187-203

Wilhelm, M., Roscher, D., Blumendorf, M., and Albayrak, S. (2010) A Trajectory-Based Approach for Device Independent Gesture Recognition in Multimodal User Interface. *Proc. of HAID' 2010*, 197–206

Wobbrock, J. O., Wilson, A. D., Li Y. (2007) Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. *Proc. of UIST'07*, 159– 168

Wolf, C. (1986) Can people use gesture commands? *ACM SIGCHI Bulletin 18* (2), 73–74

Appendix A. Woke users statistics

#	Sex	Job	Area of activity	Age	qualification	Affinity with graphical interface (grades /10)
1	M	Student	Computer science	23	University	8
2	W	Student	Computer science	24	University	9
3	W	Student	Computer science	21	University	8
4	M	Student	Economy	22	University	6
5	M	Student	Economy	19	University	5
6	W	Manager	Management sciences	48	University	6
7	W	Manager	Management sciences	41	University	4
8	M	Manager	goods transport	53	high school	5
9	W	self-employed	/	34	high school	2
10	W	self-employed	/	49	high school	3
11	M	woke's employee	Food	24	University	7
12	M	woke's employee	Food	25	University	6
13	M	Employee	Secrataryship	32	high school	5
14	W	Employee	Medicine	36	high school	4
15	M	Employee	Medicine	37	Others	3