A Model-Based Approach for Gesture Interfaces

Lucio Davide Spano
ISTI-CNR
Via G. Moruzzi 1, 56124, Pisa, Italy
lucio.davide.spano@isti.cnr.it

ABSTRACT
The interaction technologies had substantial enhancements in later years, with the introduction of devices whose capabilities changed the way people interact with games and mobile devices. However, this change did not really affected desktop systems. Indeed, few applications are able to exploit such new interaction modalities in an effective manner.

This work envisions the application of model-based approaches for the engineering of gesture user interfaces, in order to provide the designer with a comprehensive theoretical framework for usage-centred application design. The differences between existing gesture-enabling devices will be tackled applying more general solutions for multi-device user interfaces.

Author Keywords
HCI Models, Gestures, multi-touch, natural interaction

ACM Classification Keywords
H.5.2 Information interfaces and presentation (e.g., HCI): User Interfaces.

General Terms
Design, Human Factors, Languages.

INTRODUCTION

Gesture-based videogames opened the electronic entertainment market to a wider set of users and they are currently available on all major game consoles. However, the employment of this modality on different kinds of applications is still at a research stage, even in those areas where they have already demonstrated to be useful (e.g. collaborative environments, educational and museum scenarios, etc.). The main problem is the difficulty in creating such interactive applications. In order to build a gesture system, the engineers should create their hardware and software configuration for both recognizing and managing movements. The recognition system choice is usually made before the interaction design, with a really high probability of an overall poor usability of the final application.

The objective of this work will be the application of model-based approaches for user interfaces (UIs) to the gesture modality. This will provide a theoretical background to designers that want to create effective and usable gesture interfaces, first defining the interaction that should be supported and then choosing the right recognition technology. The exploitation problem of really different recognition devices can be addressed with the more general solutions for multi-device user interfaces. In particular, in this paper will be described some preliminary results for the definition of a gesture description model, which tries to provide a new abstraction for designers that overcomes the traditional event model limitations in gesture modelling.

BACKGROUND

Gestures in natural interaction consist of movements of hands, face or other body parts that are used for communication between people, replacing or enhancing speech. Gesture interfaces emulate such kind of communication, recognizing a set of gestures an exploiting them as input for computers. Many tracking and sensing technologies have been employed in order to recognize gestures. In [4] it is possible to find a survey of the different approaches.

The release of the *Nintendo Wii*[1] in 2006 marked the leverage of gesture based interaction from research studies to the market. This game console introduced an innovative controller called *Wii Remote*. Its hardware configuration allowed the development of games that break the standard and static game pad interaction, in which the player has to stay motionless and control actions pressing buttons. The user can indeed control avatars through the remote, performing the movements that should be done by their virtual counterparts. *Sony PlayStation 3* adopted a similar controller in 2010 with the *PlayStation Move*[2].

Another option for gesture recognition is the usage of motion tracking techniques in order to recognize human movements. An example of this approach is *CamSpace*[3], a software tool that exploits webcams for turning any object into a controller. This generic approach comes at the price of loosing possible haptic feedback coming from a dedicated

[1] http://www.nintendo.com/wii
[2] http://us.playstation.com/ps3/playstation-move/
[3] http://www.camspace.com

EICS'11, June 13–16, 2011, Pisa, Italy.

device: the object is only tracked and cannot be used to send interaction outputs to the user. Another great change in the way people interact with games has been produced with the launch of *Microsoft Kinect*[4], designed for the *XBox 360* game console, which was released in November 2010.

Still in game environments, it is possible to find another device type enabling for full body movement interaction, the so-called *Dance pads*. Introduced by *Konami* with *Dance Dance Revolution*[5], they are essentially huge directional pads with big arrow-shaped buttons that can be pressed with feet. This configuration allows the player to move following the music and the button sequence displayed on screen. Another kind of floor device is the *Wii balance board*, that is a rectangular feet panel that is equipped with two pressure sensors. It is mainly used in snowboard emulation games and also in aerobic and yoga activities.

GESTURE CONCRETE USER INTERFACE

In order to tackle with the existence of all these different devices and recognition capabilities, we propose to place the gesture UI modelling in the broader scope of model-based approaches for user interfaces. The gesture modality will be integrated as a *platform* into the MARIA XML [6] modelling language. A platform is a hardware and software configuration that allows the interaction with a system. The approach followed in MARIA XML is the definition of a multi-device user interface through the following levels of abstraction: the *Concepts and Tasks level*, the *Abstract User Interface* (AUI), the *Concrete User Interface* (CUI), and the *Final User Interface* (FUI). The Concepts and Tasks level contains the description of the concepts managed by the application together with the tasks that should be supported. The AUI contains a user interface description independent with respect to the device and the interaction modality. The CUI contains a user interface description abstract with respect to the technology used for the implementation. The FUI contains the final implementation of the user interface, expressed in source code.

Considering these abstraction levels, it will be defined a CUI meta-model, exploiting the concepts provided by CTT [5] for tasks and the current MARIA XML *Abstract User Interface* meta-models. More precisely, the gestures will be categorized under the *Selection*, *Control* and *Edit* interaction semantics categories. The *Only Output* interactor description will be described reusing the graphical desktop model already contained in MARIA XML. It is assumed that the screen is the main output device, together with the audio modality. Differently from the existing MARIA XML CUIs, the gesture CUI will also take into account the haptic feedback, which is a peculiar characteristic of some gesture enabling devices. The gesture CUI meta-model will be constructed using the following building blocks:

- *Gesture description model.*
- *Gesture effect model.*
- *Presentation model.*
- *Dialog model.*

The *Gesture description model* will contain the description of the gestures that can be performed for interacting with the application. The description will define only how human beings can execute a gesture, without associate any meaning to it. It will be defined a taxonomy of basic body gestures. A starting point can be the *HamNoSys*[7] notation, using the *SiGML* [2] syntax. Furthermore we need also to define the gesture composition operators, in order to allow the design of complex gestures starting from basic ones. For instance it is possible to describe the usual head movement for saying "no" with a sequence of four basic movements, starting from the head rest position: left rotation, return to the rest position, right rotation, return to the rest position.

From the human communication theory, we know that the gesture execution can be described using three categories [3, p. 86]: *static* (gesture that does not take movements into account), *dynamic* (hand trajectories or change of posture over time) and *spatio-temporal* (a subset of the dynamic gestures that move through the workspace over time). Thus the gesture description taxonomy should not only consider a sequence of body positions, but also the timing and the space used in order to perform the gesture, which can be exploited by the designer as arguments. For instance, in a golf game, the swing speed can be associated to the power of the simulated strike. The classical event model that supports the *point and click* interaction is not suitable for describing such gestures. Events are usually atomic notification of changes in some observed variables, which should be handled without assuming any temporal relation between them. Currently, gestures are modelled with single atomic events, which notify their occurrence. The problem with this notification mechanism is the lack of feedback during the performance, which is really often needed (gestures have a higher time duration compared with clicks). The solution is typically the access to the low level signals coming from input devices, with the consequent loss of an high-level event view for the considered gesture.

With the envisioned gesture description model it will be possible to manage the gesture as a whole, but there will be also the possibility to access to its components, in order to provide intermediate feedback once some basic gestures have been recognized. Moreover, it should be also possible to study the creation of an ergonomic measurement function. Indeed, once identified the basic gestures, each one can be assigned to a fatigue value. After that it can be investigated how the composition operators have influence in the perceived fatigue, with the possibility of estimating the physical cost of a complex gesture. This fatigue model should also consider the impact of the spatio and temporal characteristic of the described gesture. The gesture taxonomy and the composition operators identified can drive the creation of gesture recognizers: complex gesture recognizer can be built using a composition of basic recognizer implemented with a device-dependent library. Designers have not to reinvent the wheel for each UI, so it should be available a set frequently-

[4] http://www.xbox.com/en-US/kinect
[5] http://www.konami.com/ddr/

used complex gestures. Their definition should be ready out of the box, and they should be classified under the MARIA XML AUI interactor categories (*Select, Control, Edit*).

The *Gesture effect model* will contain the definition of the effects that the gesture interaction produces on the current UI state, together with user feedback for the gesture recognition (including also the haptic modality). This part will reuse the existing constructs contained into MARIA XML, introducing some refinements if needed.

The gesture modality usually cooperates with other ones, especially graphical and audio, because it is rarely able to provide complex feedback. For this reason, it has also to be included a *Presentation model*, which will define the graphical presentations of system output, together with audio and video media. The constructs for modelling such presentations are already defined into MARIA XML graphical CUI. If the current vocabulary is not expressive enough, such constructs will be refined. In particular, it will be considered the introduction of an explicit expression of the well-known *CARE*[1] properties for multimodal interfaces. Assigning such properties not only to graphical UI parts, but also to the gesture definitions can enable the automatized derivation of different versions of the same gesture CUI, in order to support different recognizing technologies, and also for defining different gestures with the same effect. For instance, the aforementioned head rotation gesture can be performed either starting from the right or from the left. From a descriptive point of view these gestures are different, but from the interaction point of view they can be defined as *equivalent*. These gestures can be commonly complemented with a forefinger oscillation, in this case we have the *redundancy*.

The *Dialog model* will contain the constructs for defining the complex gesture availability according to the different UI states. It will define which gestures can be used in a given state and their temporal relations (which ones should be performed in sequence, which ones can be performed in parallel etc.), the assignment of gestures to effects and the definition of the state transitions. The temporal relations between gestures can be defined using CTT [5] operators.

The depicted CUI meta-model should be able to describe gesture interfaces taking into account an abstract technology that has all the recognizing capabilities currently available. However, as should be clear from the discussion in the Background section, different technologies with different recognizing capabilities exist. Unfortunately, the differences between these devices cannot be reduced to a simple change of vocabulary between different implementation technologies (e.g. two different widget tookits for the graphical desktop platform). This means that aside the gesture CUI meta-model, it is necessary also to describe in a coherent manner the recognizing capabilities of a given set of devices. The distinction that can be currently made is between *remote based*, *motion-tracking based* and *floor device based*. It is also possible to add to these categories the *glove based*, due to the experiments that can be found in literature. These sub-platforms should be described with a *Gesture recognizer model* that should list the set of basic gestures that the subplatform is able to recognize and the composition operators supported.

PRELIMINARY RESULTS

The work for the gesture CUI definition has started with the application of the ideas described in the previous section to multitouch, which can be considered the simplest gesture interaction example. The proposed gesture description metamodel consists of three main entities: *Sample*, *Block* and *Gesture*.

Each *Sample* contains an identifier, a point, a type (start, move, end) and a flag representing its state (consumed if its data has been exploited or not consumed otherwise). It models the information provided by the device about the on-screen touches.

A *Block* represents an atomic gesture that can be recognized from samples and performs updates on gesture state. The blocks defined for multi-touch gestures are *TouchStart*, *TouchMove* and *TouchEnd*, which recognize respectively new touches, the location change of an existing touch, and the end of an existing touch (finger lifted from the screen). Each block reads the information contained in samples and optionally updates its internal state and/or the gesture-global state. When reading the information from a sample, a block can switch to a *completed* state if the recognition finished correctly or to an *error* state, if the recognition was not successful. Blocks can share a sample identifier in order to consume only samples having the same id and ignore the others (without producing errors). This mechanism links together blocks that deal with the same touch during the described gesture sequence.

A *Gesture* contains a set B of *Blocks* and their connections. $S \subset B$ contains the blocks that represent the initial gesture state (that is the atomic action to be recognized), $C \subset B$ contains the current state and $F \subset B$ contains the final states. When the recognition starts $C \equiv S$ and the gesture is considered recognized if a block $b \in F \cap C$ is completed (the atomic action that represents has been recognized). When an intermediate block completes, the gesture-defined block connections are exploited in order to add elements to C. Instead, if a block in C do not recognize its atomic action, the gesture is reset to its initial state. The gesture state contains also the data related to the current touches, which is updated by its inner blocks. When the device raises a touch-related event, the gesture forwards it only to blocks in C. When the block consumes a sample, it raises an event. This model allows the definition of gestures in four simple steps:

1. Define the set of blocks needed, marking the initial and final ones.
2. Define block connections. This specifies the gesture sequencing.
3. Define block shared identifiers, in order to assign the same touch to different blocks.
4. Define sample consumed event handlers for blocks.

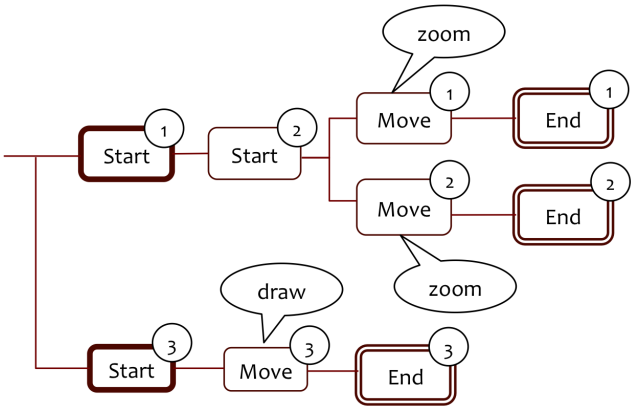

Figure 1. Prototype gesture description model

The first three steps are simply the gesture sequencing declaration, which identifies how many touches are considered and how the user will perform the interaction. Step 4 defines the interaction effect on the application data.

Figure 1 shows the multitouch gesture description modelled for the implementation of a zoomable finger-drawing canvas for the iPhone. The application recognizes two gestures, allowing the user to draw lines on a canvas with his/her finger, and to zoom the view with the classical pinch gesture. In Figure 1, rounded rectangles represent blocks. Starting ones have a bolder line while double lines represent final. The circle-enclosed numbers are touch identifiers, while balloons represent event handlers for consumed samples, attached to blocks. Lines between blocks define connections. The drawing gesture is a simple pan (Figure 1, lower part), represented by a pipe of a *TouchStart*, a *TouchMove* and a *TouchEnd* block. Each block shares the same sample identifier. It is worth pointing out that the identifiers are not related to touch ordering: having the identifier number 3 does not mean that the touch has to come after the number 1 and 2, but simply that it is different from them. The move block has a sample consumed event handler that draws a line corresponding to touch point Δx and Δy.

The pinch has been modelled as a two-finger gesture (Figure 1, higher part), which starts with the first finger touch, continues with a second finger touch. After that there is a parallel moving of the first and the second finger, and finally the end of one of the two touches. The event handler is the same for both move blocks: it evaluates the distance between the touch number 1 and 2 and updates the view zoom accordingly.

While the interaction described is not really new, there are some characteristics that should be pointed out. First of all, the interaction is completely defined by the designer composing building blocks. This means that it is possible to define custom gestures, without reimplementing the touch tracking. The designer has no more a single recognition event, but it can enhance the gesture description with handlers that provide intermediate feedback to the user, improving the interface usability. For instance, the designer can decide to draw two arrows corresponding to touches while performing the pinch gesture. These arrows converge for the zoom in or diverge for the zoom out. This is typically not possible with the high-level pinch zoom event: the application receives the notification only when the gesture is performed, without intermediate steps. Currently, the only solution is to track low level touch events. Last but not least, it is possible to express parallel gesture interaction in a straightforward manner, which is simply declaring that the two gestures work on different touches. Using the aforementioned configuration of the touch identifiers, it is possible with the prototype to draw lines while zooming the view and vice versa.

CONCLUSIONS AND FUTURE WORK

This work proposed the application of model based approaches for user interfaces to gesture interaction, in order to overcome two main problems that currently affect such modality. The first one is the existence of different recognition device with different capabilities that currently drive the interaction design. The second one is the lack of a suitable gesture definition meta-model, able to overcome the limitation of handling atomic events related to high-level gestures. A description model has been defined for multi-touch interaction, which will be enhanced in the future for body gestures and included as a platform for the MARIA XML language.

REFERENCES

1. J. Coutaz, L. Nigay, D. Salber, A. Blandford, J. May, and R. Young. Four easy pieces for assessing the usability of multimodal interaction: the CARE properties. In *Proceedings of INTERACT*, volume 95, pages 115–120, 1995.

2. R. Kennaway. Experience with and requirements for a gesture description language for synthetic animation. *Gesture-Based Communication in Human-Computer Interaction*, 2915:449–450, 2004.

3. P. Kortum. *HCI Beyond the GUI: Design for Haptic, Speech, Olfactory, and Other Nontraditional Interfaces (Interactive Technologies)*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2008.

4. S. Mitra and T. Acharya. Gesture recognition: A survey. *IEEE Transactions on Systems, Man and Cybernetics - PART C*, 37(3):311–324, 2007.

5. F. Paternò. *Model-based design and evaluation of interactive applications*. Applied computing. Springer, 2000.

6. F. Paternò, C. Santoro, and L. D. Spano. Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Transaction in Computer-Human Interaction*, 16:19:1–19:30, November 2009.

7. S. Prillwitz, R. Leven, H. Zienert, T. Hanke, and J. Henning. *HamNoSys: Hamburg Notation System for Sign Languages: an Introductory Guide*. Signum Press, 1989.