A Notation and a Layered Architecture to Model Dynamic Instantiation of Input Devices and Interaction Techniques: Application to Multi-Touch Interactions

Arnaud Hamon[1,2]**, Eric Barboni**[1]**, Philippe Palanque**[1]**, Raphaël André**[2]

[1]ICS-IRIT, University Toulouse 3,
118, route de Narbonne,
31062 Toulouse Cedex 9, France
{lastname}@irit.fr

[2]AIRBUS Operations
316 route de Bayonne
31060 Toulouse cedex 9
{Firstname.Lastname}@airbus.com

ABSTRACT

Representing the behavior of multi-touch interactive systems in a complete, concise and non-ambiguous way is still a challenge for formal description techniques. Indeed, multi-touch interactive systems embed specific constraints that are either cumbersome or impossible to capture with classical formal description techniques. This is due to both the idiosyncratic nature of multi-touch technology (e.g. the fact that each finger represents an input device and that gestures are directly performed on the surface without an additional instrument) and the high dynamicity of interactions usually encountered in this kind of systems. This paper presents a formal description technique able to model multi-touch interactive systems. A layered architecture is also proposed that proposes a generic structure for organizing models of multi-touch systems. We focus the presentation on how to represent the dynamic instantiation of input devices (i.e. finger) and how they can then be exploited dynamically to offer a multiplicity of interaction techniques which are also dynamically instantiated.

Author Keywords

Multi-touch interactions, model-based approaches, formal description techniques

ACM Classification Keywords

D.2.2 [Software] Design Tools and Techniques - Computer-aided software engineering (CASE), H.5.2 [Information Interfaces]: User Interfaces - Interaction styles.

INTRODUCTION

Over the last decade the field of interactive systems engineering had to face multiple challenges at a pace never encountered before. Indeed, while new interaction techniques have been proposed on a regular basis by the research community (e.g. multimodal gesture+voice interactions by R. Bolt in [5], post-WIMP interactions such as [4] …) recent years have seen the adoption and deployment of such interaction techniques in many different types of systems. Together with this evolution of interaction techniques, the appearance and adoption of new input devices is also a significant change with respect to the past. Indeed, mass market computers remained for nearly 20 years equipped with standard mouse and keyboard while nowadays, one interacts with more sophisticated input devices such as multi-touch surfaces, Kinect, Wiimote, …

However, these new input devices and their associated interaction techniques have significantly increased the the development complexity of interactive systems. For instance, multimodal interaction techniques are now common both as input and output modalities. One of the most challenging examples is the one of multi-touch systems[1]. Indeed, even though some studies [4] show that they improve the bandwidth between the users and the system, they bring specific challenges such as handling dynamic management of input devices (the fingers) and their associated interaction techniques (including fusion and fission of input (e.g. input fusion for a pinch) as well as fusion and fission of rendering (e.g. output fusion for fingers clustering)).

This paper first presents a formal description technique able to describe in a complete and unambiguous way the behavior of multi-touch systems. As it consists in extensions of previous work, we make explicit the changes that have been made to the ICO notation. We present the basic constructs of the extensions and how they can be applied on a simple example making particularly explicit how dynamic management of both input devices and interaction techniques are accounted for. This paper addresses more specifically multi-touch input devices and interaction techniques but the concepts are applicable to any interactive system where input devices are connected and

EGMI 2014, 1st International Workshop on Engineering Gestures for Multimodal Interfaces, June 17 2014, Rome, Italy. Copyright 2014 for the individual papers by the papers' authors. Copying permitted only for private and academic purposes. This volume is published and copyrighted by its editors.
http://ceur-ws.org/Vol-1190/.

[1] We use in this paper multi-touch systems as a shortcut for interactive systems offering multi-touch interactions

disconnected at runtime and requiring reconfiguration of interaction techniques. Secondly, the paper presents a layered architecture that structures models of multi-touch systems.

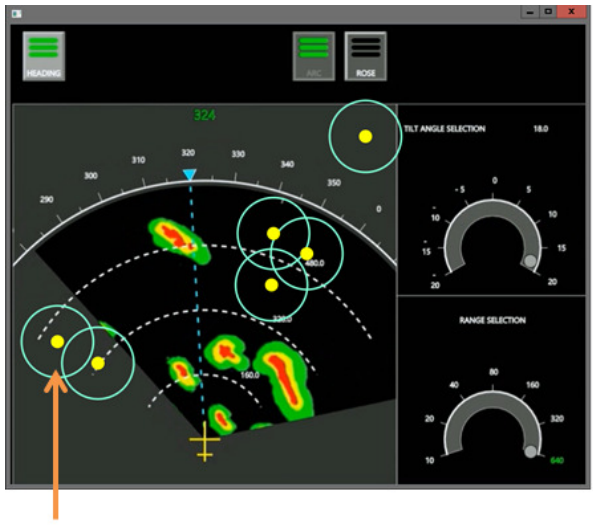
MODELLING CHALLENGES DUE TO DYNAMIC ASPECTS OF MULTITOUCH SYSTEMS

In classical interactive systems, the set of input and output devices are identified at design time and the interaction techniques to be used for interacting with the application are based on this predefined set and also defined beforehand [3]. Multi-touch systems challenge this by requiring the capacity for handling input devices (i.e. fingers) that may appear and disappear dynamically while the interaction takes place.

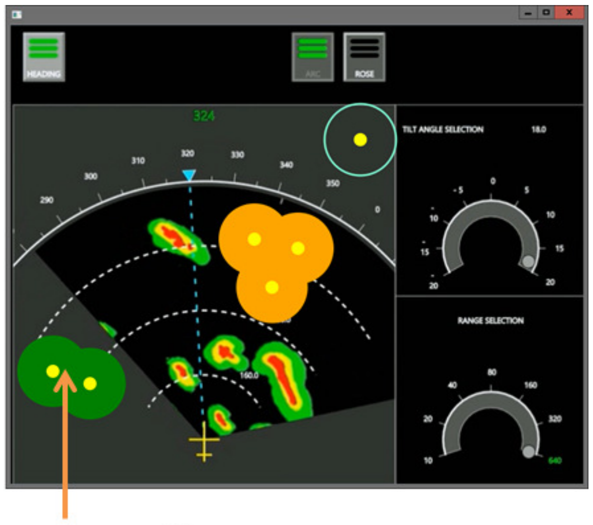
In such context, when the interactive system is started input devices are not present and thus not identified. Users' fingers are considered as input devices and are only detected as they touch (or get close enough to) the tactile surface. The input devices (fingers) detected at execution time need to be dynamically instantiated in order to be registered and listened to. While this can be easily managed using programming languages, such aspect is usually not addressed by modelling techniques in the literature. While model-based approaches provide well identified benefits such as abstract description, possible reasoning about models, complete and unambiguous descriptions, in order to deal with multi-touch systems they have to address the following challenges:

- ➔ Describe the dynamic management of input devices. This includes the description (inside models) of dynamic creation (instantiation) of input devices and the description of how many of them are present at any time. This management also requires the removal of the devices from the models when they are freed;
- ➔ Make explicit in the models the connection between the hardware (input devices) and their software counterpart (i.e. device drivers and transducers as introduced in [6] and formalized in [1]);
- ➔ Describe the set of states, the events produced and the event consumed by the device drivers and the transducers;
- ➔ Describe the interaction techniques that have to handle references to dynamically instantiated models related to the input devices (drivers and transducers);
- ➔ Describe how interaction techniques behavior evolves according to the addition and removal of input devices. Such capability is extremely demanding on the specification techniques requiring dynamic management of interaction techniques as demonstrated in [13].
- ➔ Described fusion and fission of input and output within the interaction technique. Indeed, the use of multiple input devices (fingers) makes it possible for interaction designers to define very sophisticated interaction techniques making use of several fingers grouped together for instance. Such grouping requires fusions of events from the groups of fingers but also the fusion of output information to provide feedback to the users about the current state of recognition of the interaction. For example, interaction techniques featuring a group of two fingers will require modifying the initial rendering of each finger's graphical feedback as in Figure 1-b). Figure 1-a) presents a graphical feedback of three fingers on a multi-touch application.

These challenges go beyond the ones brought by multimodal interactions identified in [12].



a) Non-clustered fingers



b) Clustered fingers

Figure 1- a) 3 input device detected; b) output of the clustering of two input devices (merged disks bottom left)

THE EXTENDED ICO NOTATION

Based on the study of the related work and the dimensions described in [9], only the ICO notation allows the explicit modelling of all the multi-touch characteristics. However, extensive modelling of multi-touch systems has demonstrated the need for modifying the ICO notation in order to provide primitives for handling specificities of multi-touch systems. It is important to note that these primitives do not constitute extensions to the expressive power of ICOs but bring the formal description technique closer to what is needed to model multi-touch systems. This is why the proposed extensions contribute beyond ICOs as such extensions could be added to other notations, provided their expressive power is sufficient for modeling multi-touch systems.

Introduction

The ICO notation (Interactive Cooperative Objects) is a formal description technique devoted to specify interactive systems. Using high-level Petri nets [8] for dynamic behavior description, the notation also relies on object-oriented approach (dynamic instantiation, classification, encapsulation, inheritance and client/server relationships) to describe the structural or static aspects of systems.

The ICO notation is based on a behavioral description of the interactive system using the Cooperative objects formalism that describes how the object reacts to external stimuli according to its inner state. This behavior, called the Object Control Structure (ObCS) is described by means of Object Petri Net (OPN). An ObCS can have multiple places and transitions that are linked with arcs as with standard Petri nets. As an extension to these standard arcs, ICO allows using test arcs and inhibitor arcs. Each place has an initial marking (represented by one or several tokens in the place) describing the initial state of the system. As the paper mainly focuses on behavioral aspects, we do not describe them further (more can be found in [14].

ICO notation objects are composed of four components: a cooperative object for the behavior description, a presentation part (i.e. Graphical Interface), and two functions (activation and rendering) describing the links between the cooperative object and the presentation part.

ICOs have been used for various types of multi-modal interfaces [11] and in particular for multi-touch [9]. This notation is also currently applied for formal specification in the fields of Air Traffic Control interactive applications [14], space command and control ground systems [15], or interactive military [2] or civil cockpits [1].

Block	Field Name	Field Description
1: Name block	name	unique name, not necessary linked to the eventName
2 : Precondition block	precondition	boolean expression independent of the event but depending on marking
3 : Event block	eventName	name of the event the transition is linked to
	eventSource	the source of the event received
	eventParameters	The collection of the parameters of the received event
	eventCondition	boolean expression based on the eventParameters' values used for the firing
4 : Action block	action	an action

Table 1- Properties of the generic event transition

Informal description of dynamic instantiation

ICOs, due to their Petri nets underpinning, are particularly efficient to create and destroy elements when they are represented as tokens. As ICOs' tokens refer to objects or other ICOs, it is possible to use such high-level tokens to represent input devices such as fingers on a touchscreen. Such tokens refer to other ICO models describing the detailed behavior of the input device. For instance, Figure 4 presents the behavior of a finger both in terms of states (values for position, pressure, ...) and events (e.g. update corresponding to move events).

The ICO model in Figure 3 describes how new input devices are instantiated and stored in a manager. The top-left transition in Figure 3 illustrates how new input devices can be added to an ICO model with the creation of a model of finger type (instruction finger=create Finger(touchinfo)). The newly created reference is then stored in a waiting place (called ToAddFinger) in order to be connected to an interaction technique in charge of handling the events that will be produced by the new device.

Handling events from dynamically instantiated sources

An ICO model may act as an event handler for events emitted by other models or java instances. The detailed description of these mechanisms is available in [16]. In addition, the different transition blocks of Figure 3 (top-left transition) are presented in Table 1.

Formal description

Due to space constraints, the formal definition of the extensions is not given here but its denotational semantics is given in terms of "standard" ICOs as defined in [14].

A LAYERED APPLICATION TO SUPPORT DYNAMIC HANDLING OF INPUT DEVICES

This section proposes a layered architecture (see Figure 2) making explicit the various models needed to describe multi-touch systems as well as the way they communicate. This architecture allows handling the dynamicity aspects of input devices and interaction techniques. The proposed architecture features "good" properties of software systems mentioned in [17] (flexibility, separation of concerns, extensibility and hardware independence). Our proposed architecture is similar to the layered based architecture described in [7] but explicitly describes the dynamic aspects related to multi-touch such as dynamic instantiation.

Figure 2 presents this architecture starting (bottom) making explicit the flow of events from the hardware (lower level) to the multi-touch interactive application (higher level). The architecture relies on existing coded layers and ICO models and is OS independent. The hardware layer describes the hardware equipment providing tactile input. In our case, this layer relates to the touchscreens considered. The hardware driver layer refers to the drivers used by the operating system and produces low-level events (such as "downs" with their basic attributes (posX, posY, …)) that propagate the upper levels. On top of this layer, the JavaFX layer provides object oriented events through an OS independent layer.

The low-level transducer (bigger box in Figure 2) is in charge of producing high-level events corresponding to the interaction techniques recognized by the system. This layer is modelled using ICO and manages the dynamicity of the input devices. The detailed description of this layer and its components is described in the followings sections.

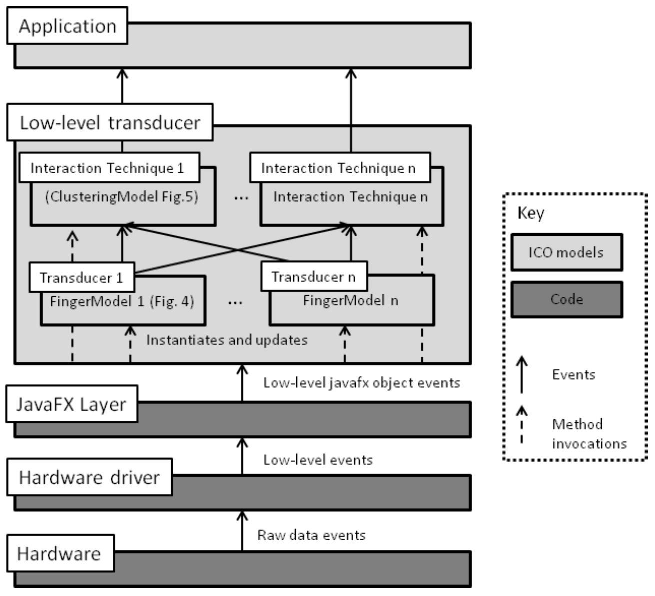


Figure 2 - Layered architecture to support dynamic handling of input devices

DEMONSTRATING HANDLING OF INPUT DEVICES: A SIMPLE EXAMPLE USING ICOS

This paragraph describes the ICO models used for the example presented Figure 1-b, which handles dynamically referenced input devices and corresponds to the main components of the architecture presented Figure 2.

Low-level transducer description

The model presented in Figure 3 is called a transducer as it is located (in terms of software architecture) in between the hardware devices and the interaction techniques as illustrated in Figure 2. There could be a chain of such models handling events from the lower level (raw events or data from the hardware input devices) to high-level events as a double click (see [1] for more details on transducers).

The low-level transducer encapsulates the references towards the upper-level models of the handling mechanism such as FingerModels and the interaction technique ClusteringModel. The role of this low-level transducer is to forward events received from the hardware to low-level events in FingerModels (which model the fingers' behavior).

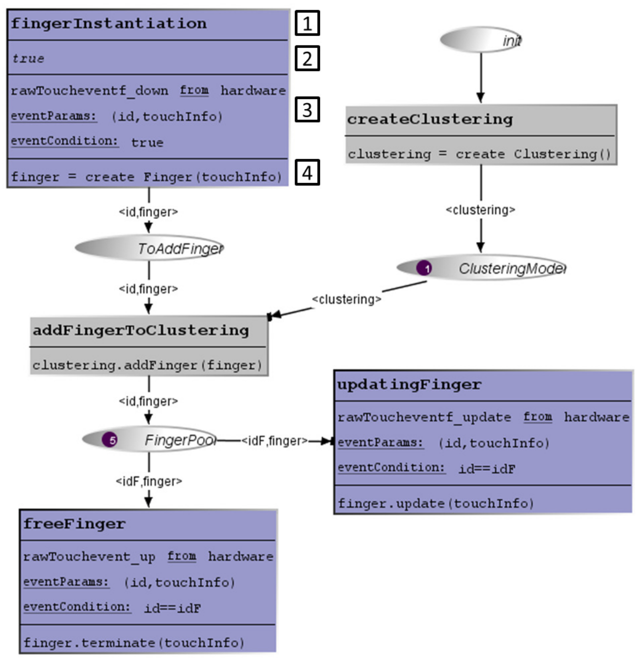
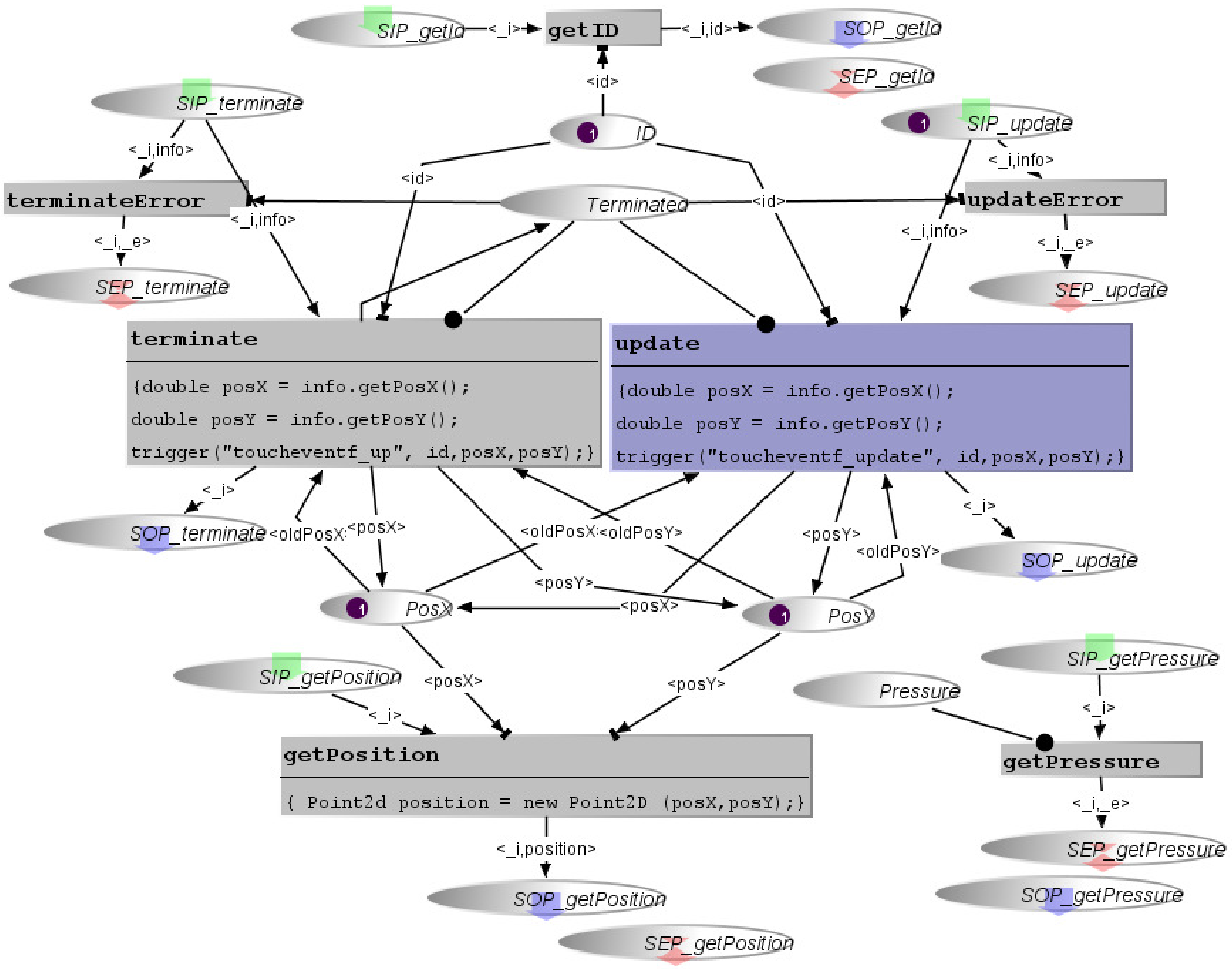

fingerInstantiation
true
rawToucheventf_down from hardware
eventParams: (id,touchInfo)
eventCondition: true
finger = create Finger(touchInfo)
1
2
3
4
init
createClustering
clustering = create Clustering()
<id,finger>
<clustering>
ToAddFinger
ClusteringModel
<id,finger>
<clustering>
addFingerToClustering
clustering.addFinger(finger)
<id,finger>
FingerPool
<idF,finger>
<idF,finger>
updatingFinger
rawToucheventf_update from hardware
eventParams: (id,touchInfo)
eventCondition: id==idF
finger.update(touchInfo)
freeFinger
rawTouchevent_up from hardware
eventParams: (id,touchInfo)
eventCondition: id==idF
finger.terminate(touchInfo)

Figure 3 – Excerpt of the model of a low level transducer

During the initialization, the low-level transducer instantiates the ClusteringModel through the createClustering transition and stores its reference in the ClusteringModel place. When the low-level transducer receives a "rawToucheventf_down" event from the hardware, the fingerInstantiation transition is fired, the event parameters (the touch iD, and its additional information) are retrieved and used to dynamically instantiate a new instance of FingerModel. The addFingerToClustering transition then adds the FingerModel reference to the cluster model. This is how the interaction technique is informed of the detection of new fingers. The low-level transducer then stores the reference of the FingerModel in the FingerPool place (which contains the list of all the detected fingers). When the transducer receives "rawToucheventf_update" (resp. "rawToucheventf_up") events from the hardware, the transition updatingFinger (resp. freeFinger) is then triggered and updates accordingly the proper FingerModel. These updates are provided using the communication mechanism of ICO services and not using events since the low-level transducer contains references toward the FingerModels and is able to match the hardware events with the right model.

described offering various properties such as finger tilt angle, acceleration and direction of the movements.

Lastly, this finger model is an extensible model that can describe very complex behaviors. For example, if one needs to describe the behavior of a finger input as in Proton++ [10], this can be done in a finger model as the one presented. Indeed this model specifies when the touch



```
{double posX = info.getPosX();
double posY = info.getPosY();
trigger("toucheventf_up", id,posX,posY);}
```

```
{double posX = info.getPosX();
double posY = info.getPosY();
trigger("toucheventf_update", id,posX,posY);}
```

```
{ Point2d position = new Point2D (posX,posY);}
```

Figure 4 – Generic Model of Finger

Modelling touch fingers

Each time the low-level transducer receives an event corresponding to the detection of a finger on the hardware, it creates the model and links it with the interaction technique model(s). When the event received corresponds to an update of an already detected finger, the low-level transducer notifies the corresponding finger model using the services "update". When the finger is removed from the hardware, the low-level transducer fires the transition freeFinger, which destroys the corresponding FingerModel.

For readability purposes, the model presented in Figure 4 features a limited set of fingers properties: position and pressure. However, more complex finger models have been

events are broadcasted and that such broadcasting can be controlled in order to match a sequential system sending user events every 30ms as in [10].

Modelling the interaction technique "finger clustering"

This paragraph describes how the ICO notation handles interaction techniques including output fusion of information related to the reception of events produced by dynamically instantiated input devices (see Figure 5). In this example, the interaction technique model is in charge of pairing co-located input devices so they can be handled as a group of fingers. This corresponds to the interaction presented in Figure 1 where the right-hand side of the figure presents the rendering associated to the detection of a pair

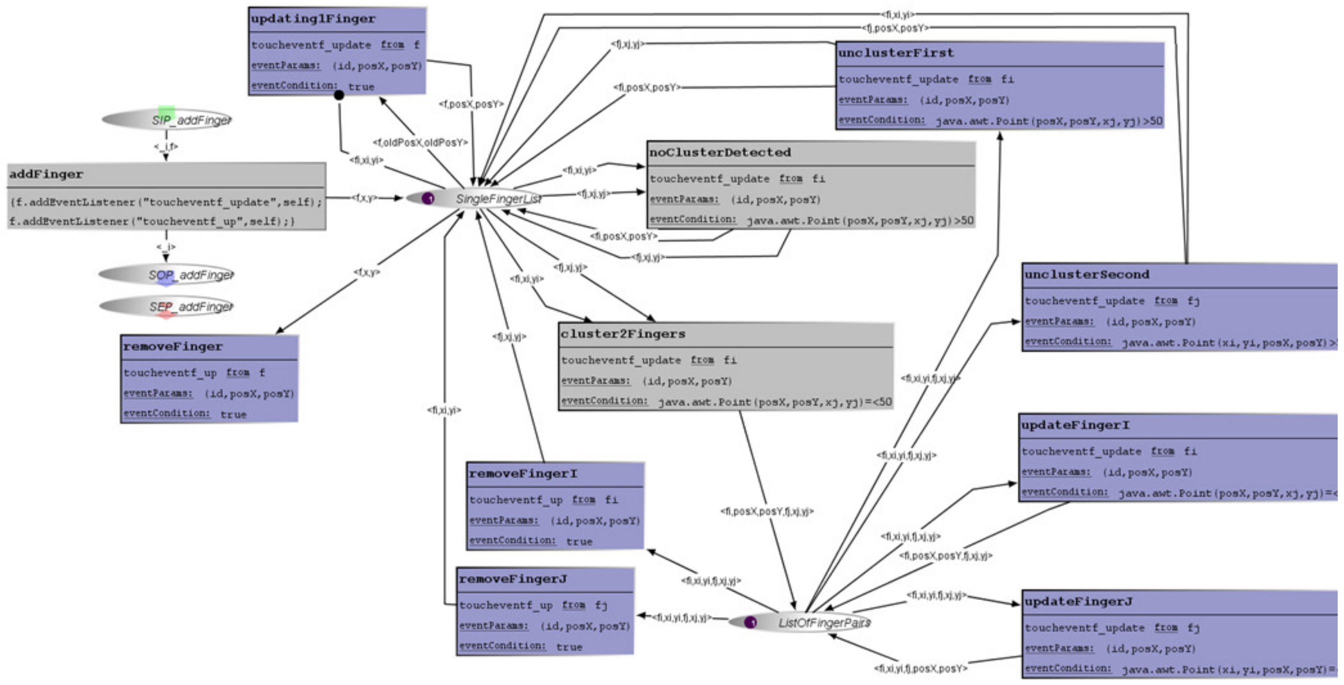


Figure 5 - Model of the interaction technique “finger clustering”

of fingers (bottom-left of the figure) while the other finger remains ungrouped. The model presented in Figure 5 is composed of a service (addFinger), two places (ListOfPairs storing the pairs of fingers and SingleFingersList storing the “single” fingers) and event-transitions to update the clustering according to the evolution of the position of fingers on the touchscreen. Each time a finger model is created (a new finger touches the screen), the low level transducer calls the “addFinger” service and a reference to a new finger model is set in place SingleFinger. When a finger from SingleFingerList (called finger1 for instance) moves close enough to another finger (e.g. finger2) in that place too, two cases are represented:

- finger2 is close enough of finger1 (condition in the event condition zone of transition cluster2Fingers is true) then transition cluster2Fingers is fired, finger1 and finger2 are removed from place SingleFingerList and a new token consisting of the pair (finger1, finger2) and their respective position is stored in place ListOfFingerPairs.
- finger2 is too far from finger1 (condition in the event condition zone of transition noClusterDetected is true) then that transition is fired and the new position of finger is updated.
- When a pair is detected, the user interface should display graphically such dynamic grouping. This is defined by the rendering function associated to the interaction technique and presented in Table 2. When two fingers are merged, the token referencing these two models are removed from SingleFingerList place which triggers the method hideFingerRendering for each model. This method hides the elementary rendering associated to each finger. When a pair is detected, both references are combined in a token added to the place LisfOfFingerPairs which calls the method createPairedFingerRendering, which displays the rendering associated to the two-finger cluster.It is important to note that output is thus connected to state changes in the models (which only occur when tokens are added to or removed from places) while inputs are event based and thus associated to transitions.

ObCS Node name	ObCS event	Rendering method
SingleFingerList	tokenAdded	showFingerRendering
SingleFingerList	tokenRemoved	hideFingerRendering
ListOfFingerPairs	tokenAdded	createPairedFingerRendering
ListOfFingerPairs	tokenRemoved	removePairedFingerRendering

Table 2 -Rendering functions of the interaction technique

CONCLUSION

This paper has identified a set of challenges towards the production of complete and unambiguous specifications of multi-touch systems. The main issues deal with the dynamic instantiation of input devices and the dynamic reconfiguration of interaction techniques. We have highlighted the fact that such concerns have not previously encountered (at least at this large scale) when engineering interactive systems. This paper has presented a twofold way for addressing these issues:

- A layered software architecture made up of communicating models, which makes explicit a set of components and their inter-relations in order to address this dynamicity challenge;
- A formal description technique able to describe in a complete and unambiguous way such dynamic behaviors.

While the formal notation contribution is very specific to the work presented here, the layered architecture is independent from it and can be reused within any framework dealing with multi-touch interactions.

REFERENCES

1. Accot J., Chatty S., Maury S. & Palanque P. Formal Transducers: Models of Devices and Building Bricks for Highly Interactive Systems. DSVIS 1997, Springer Verlag, pp. 234-259.
2. Bastide R., Navarre D., Palanque P., Schyn A. & Dragicevic P. A Model-Based Approach for Real-Time Embedded Multimodal Systems in Military Aircrafts. Int. Conference on Multimodal Interfaces (ICMI'04), ACM DL, 10 pages.
3. Bellik Y., Rebaï I., Machrouh E., Barzaj Y., Jacquet C., Pruvost G., Sansonnet J.-P.: Multimodal Interaction within Ambient Environments: An Exploratory Study. INTERACT (2) 2009: 89-92
4. Bi X., Grossman T., Matejka J., and Fitzmaurice G.: 2011. Magic desk: bringing multi-touch surfaces into desktop work. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11). ACM, New York, NY, USA, 2511-2520.
5. Bolt, R and Herranz, E. (1992). “Two-Handed Gesture in Multi-Modal Natural Dialog”, Proceedings of the fifth annual ACM symposium on User interface software and technology, ACM Press, p 7-14
6. Buxton W. A three-state model of graphical input, IFIP TC 13 INTERACT’90, 1990, p. 449–456.
7. Echtler F. and Klinker G.. 2008. A multitouch software architecture. In Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges (NordiCHI '08). ACM, New York, NY, USA, 463-466.
8. Genrich, H. J. 1991. Predicate/Transitions Nets. In High-Levels Petri Nets: Theory and Application. K. Jensen and G. Rozenberg, (Eds.), Springer Verlag (1991) pp. 3-43
9. Hamon A., Palanque P., Silva J-L., Deleris Y., & Barboni E. 2013. Formal description of multi-touch interactions.5th symp. on Engineering interactive computing systems (EICS '13). ACM, 207-216
10. Kenrick K., Björn H., DeRose T. & Maneesh A. 2012. Proton++: a customizable declarative multitouch framework. Proc. of ACM symposium on User interface software and technology (UIST '12). ACM, 477-486.
11. Ladry J-F., Navarre D., Palanque P. Formal description techniques to support the design, construction and evaluation of fusion engines for sure (safe, usable, reliable and evolvable) multimodal interfaces. ICMI 2009: 185-192
12. Lalanne D., Nigay L., Palanque P., Robinson P., Vanderdonckt J. & Ladry J-F. Fusion engines for multimodal input: a survey. ACM ICMI 2009: 153-160, ACM DL
13. Spano L-D., Cisternino A., Paternò F., Fenu G. GestIT: a declarative and compositional framework for multiplatform gesture definition. EICS 2013: 187-196
14. Navarre D., Palanque P., Ladry J-F. & Barboni E. ICOs: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. ACM Trans. Comput.-Hum. Interact., 16(4), 18:1–18:56. 2009
15. Palanque P., Bernhaupt R., Navarre D., Ould M. & Winckler M. Supporting Usability Evaluation of Multimodal Man-Machine Interfaces for Space Ground Segment Applications Using Petri net Based Formal Specification. Ninth Int. Conference on Space Operations, Italy, June 18-22, 2006
16. Palanque P. & Schyn A. A Model-Based Approach for Engineering Multimodal Interactive Systems in INTERACT 2003, IFIP TC 13 conf. on HCI, 10 pages.
17. Khandkar S.H. & Maurer F. A domain specific language to define gestures for multi-touch applications. In Proceedings of the 10th Workshop on Domain-Specific Modeling (DSM '10). ACM, New York, NY, USA, , Article 2 , 6 pages.