Txupito
An Interactor Component Model for Ambient HCI

Inès de Courchelle, Philippe Roose, Marc Dalmau and Patrick Etcheverry
LIUPPA Laboratory, Anglet, France

Keywords: Interaction, Interactor Model, Ambient, Widget.

Abstract: In this paper, we describe the Txupito interactor model. Nowadays, application contents are accessible on a wide variety of devices such as laptops, tablets and smartphones but the heterogeneity of such devices does not help programmers to elaborate ambient interactive applications adapted to each device. Indeed, users do not interact in the same way on a tablet, a laptop or a smartphone. Furthermore the design of cooperative HCI (Human Computer Interface) remains a very complex task especially in an ambient and heterogeneous environment. In this paper, we propose a multimodal mobile interactor model, called Txupito, and a set of composition operators as a way to achieve interactions with hardware in a distributed and cooperative way.

1 INTRODUCTION

Applications are now present in our everyday life through smartphones, laptops, and tablets with which we interact differently. To improve users' interaction experience, parts of HCI (Human Computer Interactions) need to be created or deleted on a device or moved (migrated from one device to another) at runtime. In this context, designers have to rethink the way they implement interactions in their applications because these interactions are now potentially spread over multiple devices with different interacting modalities. For example, the same application can be controlled with a keyboard and a mouse on a laptop whereas it will be voice controlled on a smartphone. In this paper, we focus on pervasive interfaces. How to share widgets in spite of heterogeneity of devices?

This raises two issues. The first one concerns the runtime environment because ours applications are executed in a mobile environment and the presence of the devices is not guaranteed due to mobile constraints (mobility, energy, etc.). The second one concerns the device itself because we cannot predict its technical specifications (size screen, embedded sensors, environment, etc.) and the way to adapt interactions when the application is deployed.

In this paper, we consider an application as composed of a set of interconnected services each one made up as a set of software components. We adapt such an application on runtime with the help of a software platform called Kalimucho (Da *et al.*, 2014). This platform manages applications based on software components linked with first class connectors. It allows to dynamically [re-]deploy components at runtime on multiple devices. It supports live migration of services as well as addition/suppression of components (ie. services).

In this article we will explain how we use this pervasive platform to solve the interactions problems previously described. In that aim we will define pervasive interactors called Txupito. Interactors are implemented as software components and deployed on multiple devices so the HCI can be adapted at runtime.

This article is organized as follows: in Section 2, we will present a scenario, which will help us to introduce and illustrate ours definitions. Section 3 briefly describes the architectural layers of considered applications. Then, in Section 4, 5 and 6, we will introduce our Txupito interactor model. Afterward in Section 7, we will present a prototype that integrates Txupito interactors. In Section 8, we will confront our work to related works in interaction and software components. Finally, we conclude this paper with suggestions on future work.

2 SCENARIO

The following scenario will be used to illustrate our approach; it applies in the case of a conference

de Courchelle I., Roose P., Dalmau M. and Etcheverry ..
Txupito - An Interactor Component Model for Ambient HCI.
DOI: 10.5220/0005226200670075
In *Proceedings of the 5th International Conference on Pervasive and Embedded Computing and Communication Systems* (PECCS-2015), pages 67-75
ISBN: 978-989-758-084-0
Copyright © 2015 SCITEPRESS (Science and Technology Publications, Lda.)

(Figure 1). A speaker presents a slideshow to an audience. To facilitate his speech, this slideshow can be directly sent to the public's smartphones.

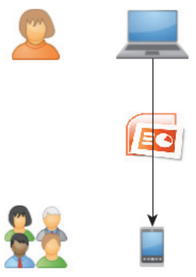


Figure 1: Scenario schema.

3 PRINCIPLES: OUR APPLICATIONS ARCHITECTURE

Applications are based on a three layers architecture (Figure 2):

- **Business Layer:** it is the functional core. It is composed of a set of components linked by connectors. This component is called a "Business component" as it encapsulates the functional part – not the interaction part;
- **Users Layer:** it concerns the users interacting possibilities with an interface. An interface is composed of a set of interactors that can be graphical or not;
- **Interactor Layer:** it represents the bridge between the business layer and the user. It lets the user manage the business tasks.

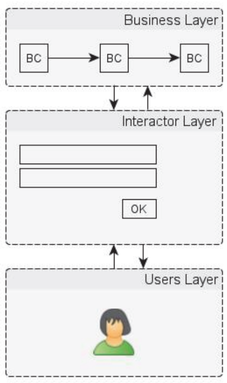


Figure 2: Application Layers.

It is important to note that in Figure 2 the Users Layer concerns all the users that interact with the application through a distributed HCI ("Interactor Layer") while the Business Components (BC) ensure processing of information. All these layers are distributed on all devices involved in the application.

4 INTERACTOR

Usually programmers use widgets to implement interactions on applications. We will instead speak of interactors in order to clearly separate the interaction from the graphical aspect.

4.1 Description

Interactors allow communication between the users and the functional core (Business Layer). They are divided into four parts (Figure 3):

- **Abstract Object (AO):** Contains the type and the value understood by the Functioning core;
- **Interpreter (I):** Converts physical information into a language understood by the system;
- **Concrete Object (CO):** Contains the type and the value produced by the physical Layer;
- **Physical (P):** Captures the type and the value produced by a physical device.

An interactor holds one or several Concrete Objects and one or several Physical parts but only one Interpreter and one Abstract Object. Interactors are divided in input and output interactors.

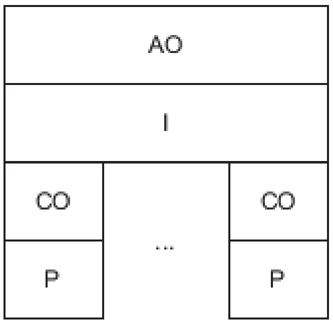


Figure 3: Interactor schema.

For example, the speaker can switch from one slide to another via a leap motion (https://www.leapmotion.com/). This is a sensor (including two infrared cams) that captures hand movements. A movement from left to right makes the next slide appears while a movement from right to left makes the previous slide appears. This interaction is represented by an input interactor. The

Physical layer is the leap motion sensor's API. It captures the movements of the speaker's hand. The Concrete Object holds the frames captured by this API. The Interpreter traduces these frames into "next" or "previous". The Abstract Object represents the “next” or “previous” data requested by the business component. In this example, the interactor has only one Concrete Object and Physical part. An interactor realized by two buttons will have two Concrete Object and two Physical parts. The first one is the “next” button, its Physical part is the Java API and its Concrete Object is the JButton known by the java API. The second one is the “previous” button, it has its own Physical part and Concrete Object. Then the Interpreter takes these two Concrete Objects to create an Abstract Object that represents the “next” or “previous” data.

4.2 Modality

A modality is how a user interacts with the system. It only concerns the Physical part, the Abstract Object and the Interpreter of an interactor. In the leap motion example, the modality is the movement. The same interactor with a different modality (mouse left or right click) will have a Physical part concerning the mouse API, its Concrete Object will be the number associated to mouse buttons and its Interpreter will traduce these numbers into "next" or "previous". The Abstract Object is the same so the Business Layer can indifferently use the leap motion or the mouse in order to allow the speaker making its slideshow (Table 1).

Table 1: Modality example.

Interactor Changes slides	With Leap Motion	With Mouse
Physical	Leap Motion API	Mouse API
Concrete Object	Frames	Button Number
Interpreter	Next or Previous based on frames	Next or Previous Based on number
Abstract Object	Next or Previous	

4.3 Interaction

We distinguish three types of interaction:

Input interaction (Figure 4) is designed by an input interactor (II). Data (the Abstract Object) can be sent to a business component. It corresponds to the previous example (with the leap motion).

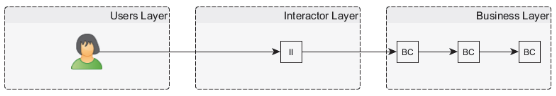


Figure 4: Input Interaction.

Output interaction (Figure 5) is designed by an output interactor (OI). Data (the Abstract Object) comes from a business component and is communicated to the user through the physical layer. Data can be represented under a graphic form, a tactile way (vibration), by a speech synthesis system, etc. For example, the display of a slide is an output interactor.

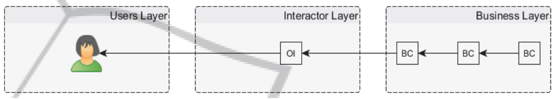


Figure 5: Output Interaction.

Input interaction with feedback (Figure 6) is designed by a combination of an input (II) and an output interactor (OI). It allows the user to get a feedback of its input interactions. A user action on the physical layer of the II produces an Abstract Object, which is directly sent to the OI that ensures the feedback. Of course, this AO can also be sent to a Business Component. For example, when the speaker wants to go to a specific slide using a voice modality, he says the number of this slide. The input interactor uses voice recognition and produces the number of the slide to show. The output interactor is a text field automatically filled with this number. The input interaction carries the voice modality. The output interaction (feedback) is the slide number shown in the text field. The Abstract Object is sent to the business component in order to change the slide.

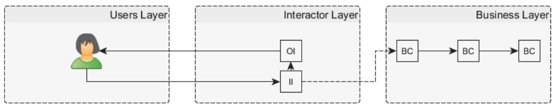


Figure 6: Input Interaction with feedback.

5 THE TXUPITO INTERACTOR MODEL AT RUNTIME

In this part, we will describe the Txupito interactor model. As Txupito is a Kalimucho component, the platform can add/remove/migrate interactors at runtime in order to adapt the HCI to the context and the wishes/needs of users.

5.1 Changing the Semantic

Obviously the semantic of an interactor is given by the Abstract Object. Changing the Abstract Object (AO) and consequently the interpreter means changing the semantic of the interactor. For example (Table 2), if we want to use the leap motion to change the sound level, the Abstract Object (AO) will concern the sound level. The Interpreter will now use the speed information of the movement detector included in the frames information. So it will extract this information from the frames data of the API in order to produce an Abstract Object (AO) of the form "strongly lower", "lower", "little lower", "little louder", "louder" and "strongly louder".

This possibility of keeping the same Physical layer and Concrete Object while changing the Interpreter and the Abstract Object allows reducing the work of programming different interactors.

Table 2: Changing the semantic.

	Interactor Volume	Interactor Change Slides
Physical	Leap Motion API	
Concrete Object	Frames	
Interpreter	Reduce or increase sound	Next or previous slide
Abstract Object	"strongly lower", "lower", "little lower", " little louder", "louder" or " strongly louder"	“next” or "previous"

5.2 Changing the Modality

Sometimes, application context leads us to change the way we interact with our applications. For example, during the conference, the speaker is stuck at his desk because he has to change the slides with the leap motion. Therefore, if he wants to move, the modality has to be changed in order to allow him controlling slide changes with buttons on his smartphone. For this new interactor, the Abstract Object will be the same but the Physical part, the Concrete Object and the Interpreter have to be changed.

5.3 Assembling Interactors

The synchronization of several input interactors, with or without feedback, is made possible by the CARE properties (See section 8.1). These properties are the following: Complementarity, Assignment, Redundancy and Equivalence. All this four properties are operators because we can assemble several interactors together thanks to them. In this part we will describe how to combine interactors with these operators.

5.3.1 Complementarity

Complementarily (Figure 7) is the use of two or more interactions to achieve one command. For example, changing the sound volume can be made through a hand movement but in order to avoid errors we can add a confirmation button. Changing the volume is now carried out by a movement above the leap and a click on a button to validate. This is achieved by an input interactor with feedback (leap motion and text field) and another input interactor (button). Abstract Objects of both interactors, are sent to a Complementary Operator that produces the final Abstract Object to the Business Layer.

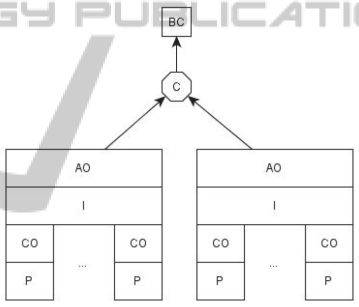


Figure 7: Complementarity Operator.

5.3.2 Assignment

Assignment (Figure 8) allows using only one interactor to achieve an action. The interface or the business component waits for a particular interaction. This operator checks if the interactor is the good one. Otherwise, the interaction is not taken into account by the system. This operator expresses

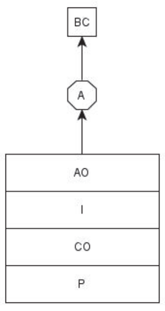


Figure 8: Assignment Operator.

the absence of choice. For example, the only way to hear the speech of the speaker is to switch on a button. The business component for sound restitution allows only one interactor.

5.3.3 Redundancy

Redundancy (Figure 9) is the use of several interactions to achieve an action. All these interactions are necessary to validate the action. They have the same interpretation and concrete object. For example, at the end of the presentation, public members can send the slide number on which they want to ask a question. The speaker has to select the same slide number to validate the demand.

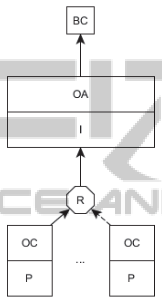


Figure 9: Redundancy Operator.

5.3.4 Equivalence

Equivalence (Figure 10) allows using several interactors to achieve the same action. For example, to change the slide the speaker can use the leap motion or buttons on his smartphone. Both produce the same Abstract Object and the operator produces the same Abstract Object each time it receives one from one interactor.

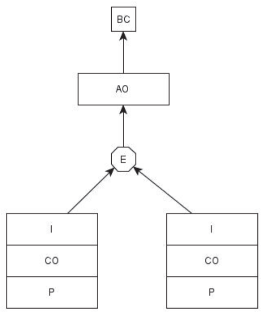


Figure 10: Equivalence Operator.

5.4 Migration

Migration is moving an interactor from one device to another. The Kalimucho platform allows this migration during execution. For example, in our scenario if the speaker's smartphone has a low battery, the interactor "next/previous" with two buttons can be moved to his laptop.

Migration is different from removing the old interactor and creating a new one because the Interpreter can be an automata and can hold a state. When a component is migrated its actual state is sent to the new host so the functioning of the Interpreter is ensured after the migration.

6 APPLICATION ORGANIZATION

Ours applications are distributed on many devices, i.e. several parts of the application run on various devices. We use a platform called Kalimucho to build and reconfigure applications made of an assembly of software components.

For example, during the conference, the platform deploys the slide show Business Components, the input interactors to control slides, the operators on the speaker's devices and the output interactors to read slides on the publics' devices. The speaker has several interactors to change slides and uses more than one device. According to the Business Components deployed on each public device, the user can see slides as normal pictures or as texts only. The application is organized as follows (Figure 11):

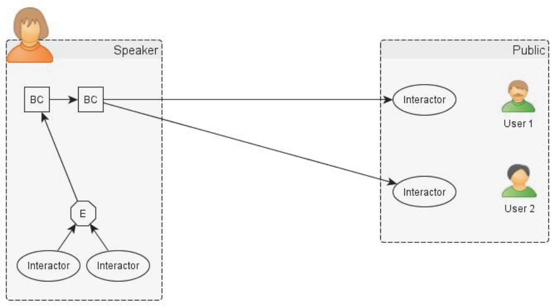


Figure 11: Application Organization.

The platform is able to choose the Business Component it deploys on each device according to the available resources on this device (Da *et al.*, 2014). So, on a Smartphone with a little screen it will show slide's text only while in a big one it will show slides as pictures.

7 PROTOTYPE

In order to validate these propositions, we developed a prototype corresponding to the illustrating example of this paper. The speaker can change slides with his keyboard, his smartphone and/or a leap motion connected to his laptop. The public see the current slide on their smartphones in two formats: text only or picture.

7.1 Snapshots

7.1.1 Speakers' HCI

Figure 12 shows the presentation displayed on the speaker's laptop. Slides appear as pictures on the screen.

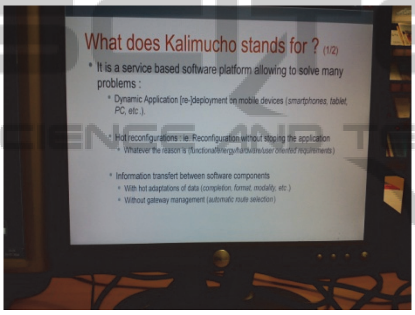


Figure 12: Speaker's laptop.

He can change the slides with a leap motion (Figure 13) using a hand movement. A movement from left to right makes the next slide appears while a movement from right to left makes the previous slide appears.

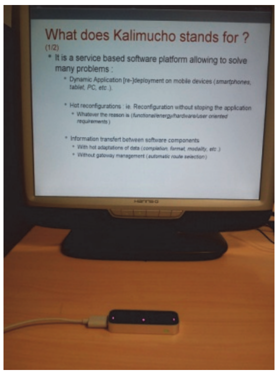


Figure 13: Speaker's Leap Motion.

Moreover, the speaker can also control the slides with his keyboard left and right arrows keys (Figure 14).

At the end, if he wants to move away from his desk, a new interactor allows him changing slides from his smart phone (Figure 15).

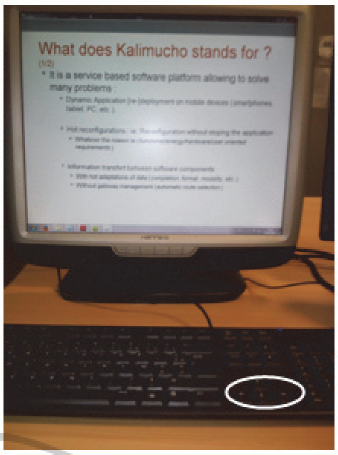


Figure 14: Speaker's keyboard.

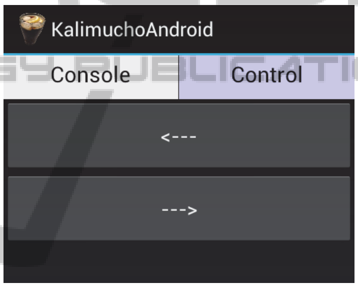
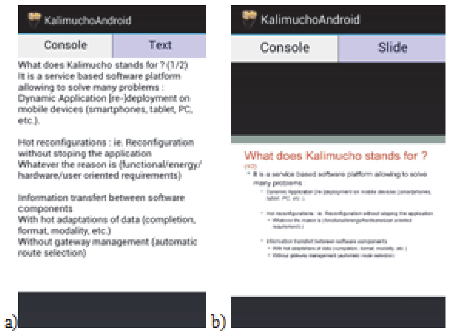


Figure 15: Speaker's smartphone.

7.1.2 Public's HCI

To follow the conference, the current slide is sent to the smart phones of the public. If the smart phone has a small screen only the text of the slide is sent (Figure 16 a). Otherwise, the picture is displayed (Figure 16 b).



What does Kalimucho stands for ? (1/2)
It is a service based software platform allowing to solve many problems :
Dynamic Application [re-]deployment on mobile devices (smartphones, tablet, PC, etc.).

Hot reconfigurations : ie. Reconfiguration without stoping the application
Whatever the reason is (functional/energy/hardware/user oriented requirements)

Information transfert between software components
With hot adaptations of data (completion, format, modality, etc.)
Without gateway management (automatic route selection)

Figure 16: Current slide text on public's smartphone.

7.2 Prototype Architecture

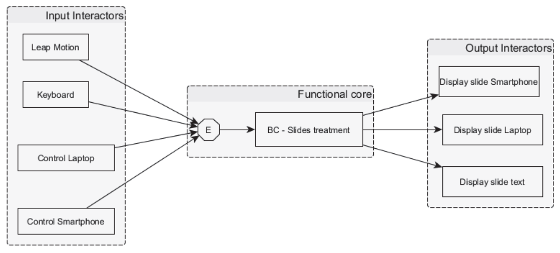


Figure 17: Application Graph.

The application graph is shown in Figure 17. The left side of the figure holds all the speaker's input interactors. They collect the Abstract Object “next” or previous” and are linked by an Equivalence operator (E). In the functional core (center side of the figure), the Abstract Object is sent to the Business component that uses it to change the slides. When a slide is selected, it is sent to the output interactors for the public members (right side of the figure) in different formats (picture for laptop, text, or picture for smartphones).

8 RELATED WORK

This section presents a state of the art related to interaction, interactors, and software components.

8.1 Interaction and Multimodality

L.Nigay (Nigay and Coutaz, 1996) describes the modality of interaction as a couple <Device, Language>:

- **Device:** physical component of the system that allows collecting information. This is the lowest level of abstraction of the modality. (for example: keyboard, mouse ...);
- **Language:** defines a set of data understood by the computer system.

Txupito also has a device and a language part. The device part is the Physical layer and the Concrete Object, the language part is the Abstract Object. The Interpreter constitutes the bridge between these two parts.

The first interface with two modalities was the "put that here" of R. Bolt (Bolt, 1980). J. Coutaz and L. Nigay (Nigay, L. and Coutaz, 1994) have structured multimodality via the CARE properties (see Section 5.3). This model aims to combine a set of modalities. These properties are the following: Complementarity, Assignment, Redundancy, and Equivalency.

Complementarity between devices or interactions languages, expresses the need of every device or interaction language to get a complete command. The use of several devices or interactions languages may be parallel or sequential. For example, the user must identify himself via a password and via his fingerprint to validate identification. He must perform both actions.

Assignment expresses the obligation to use a device for a given interaction language or a particular interaction language for a task. This property expresses the absence of choice. For example, the only way to turn a computer on is to use the button provided for this purpose.

Redundancy The use of two different interactions contributes to reliability the data collected. For example, using a microphone to capture the voice of a user and, to make the system more reliable, adding a camera which observes the motion of his lips. Those will allow increasing the robustness of the speech recognition system for example when there is noise.

Equivalency of modalities is satisfied if each device or interaction language achieves the same goal in producing the same data. For example, a computer provides a user the possibility to use the mouse. This allows him to move the elevator of a page through the widget or to use the mouse wheel. These two interactions are equivalent. User may use either one or the other.

8.2 Interactor Model

The CNUCE interactor has been introduced by Faconti et Paterno in the 90’s (Faconti and Paterno, 1990); (Paternò and Faconti, 1992) to help interaction between an HCI and a user. It is a component of the user HCI that can operate in parallel with others interactors. Moreover, they define it as a feature of an interactive system able to react to external stimuli, resulting in a high level of data abstraction to a lower abstraction level and vice-versa.

Duke and Harrisson in 1993 (Duke and Harrison, 1993), developed the York model. The difference from CNUCE, is that it is possible to model more precisely the interactive dialog system. The state of the interactor is explicitly described.

The Cert Interactor (Roché, 1998) allows the description of an interactive system with a language based on data streams.

These interactors are designed for graphical interfaces while Txupito can be used for graphical interfaces or not (tactile vibrations, speech synthesis, voice recognition, etc.).

Markopoulos (Markopoulos *et al.*, 1997); (Markopoulos 2001) defines an interactor as a primitive abstraction used in the description of an interactive system. This abstraction can be considered as similar to software architecture abstractions of objects in object-oriented programming. There is no difference between an input interactor and an output interactor. It is difficult to identify the associated modalities. For Txupito we choose to make a difference between input, output and input with feedback interactors because it is closer to the well known widget model.

Then, there is the Comet interactor introduced by (Daâssi *et al.*, 2003); (Calvary *et al.*, 2004). They define it as "an introspective interactor that publishes the quality in use it guarantees for a set of contexts of use. It is able to either self-adapt to the current context of use, or be adapted by a tier-component. It can be dynamically discarded (versus recruited) when it is unable (versus able) to cover the current context of use". Txupito is an ubiquitous interactor but the adaptation is not done on the interactor itself but through the Kalimucho platform directly on the architecture of the application. Interactors and Business Components achieving the complete applications are seen as software components linked by connectors and the reconfigurations concerns as well Interactors than Business Components. It is so possible to adapt the complete application to the context and to users' needs.

8.3 Plateforms

8.3.1 WComp

WComp (Cheung-Foo-W *et al.*, 2006) is a lightweight component-based approach for designing composite web services. It provides a framework allowing applications to be constructed in the form of web services graphs based on the Container concept. It also provides a middleware based on the concept of Aspects of Assembly used to adapt the web services. The WComp framework is based on the SLCA paradigm, *Service Lightweight Component Architecture* (Cheung-Foo-W *et al.*, 2006); (Hourdin *et al.*, 2008). This paradigm combines the principles of event-based web services paradigm and components paradigm.

8.3.2 Kalimucho

The LIUPPA (Da *et al.*, 2014); (Dibon *et al.*, 2013) creates a platform called Kalimucho. This platform allows building and structuring an application as an assembly of software components. The resulting system has properties of reusability (reusability of components), partitioning (cutting in components), optimization (assembly of components) and adaptation (architecture configuration). A component gives a specific service and can be linked to others components in order to achieve a complete user's service.

8.3.3 Why Kalimucho for Txupito?

Instead of using WComp, we choose to design the Txupito interactor for the Kalimucho platform because Kalimucho offers reconfiguration at runtime. Indeed, in order to improve users' interaction experience, parts of the HCI need to be moved, created or deleted on a device at runtime. Moreover, the platform is intrusive: it measures context changes constantly and applies needed modifications dynamically. For example, in our scenario, if the battery of the speaker's smart phones goes low, the platform automatically migrates the II for changing slides to the laptop.

9 CONCLUSION

We have presented the Txupito interactor model, its general architecture and the CARE operators to combine interactions. Txupito is an interactor for pervasive and ambient interfaces. The implementation of these interactors is similar to a software architecture made of components connected with connectors (traditional approach). It is here implemented with the help of the Kalimucho platform. This platform allows the applications to be distributed on different devices and re-configured while running. Txupito has its own architecture allowing the evolution of modalities on several devices, including operators.

Such new functionalities raise several problems – not addressed here – in term of engineering. The design of distributed HCI including the evolution of interactions needs a new approach to design scaled applications.

REFERENCES

Nigay, L., Coutaz, J., *Espaces conceptuels pour l'interaction multimédia et multimodale*, TSI, special Multimédia et Collecticiel, AFCET & Hermes Publ., Vol 15(9), pp. 1195-1225, (1996).

Bolt, R., Put that there*: Voice and gesture at the graphics interface*. Computer Graphics, p. 262-270 (1980).

Nigay, L., Coutaz, J., *Les propriétés "CARE" dans les interfaces multimodales*, lille, IHM'94 p7-14 (1994).

Faconti G, Paterno F, *An approach to the formal specification of the components of an interaction*. In C. Vandoni and D. Duce editors, Eurographics 90, pp 481-494, Noth- Holland, (1990).

Paternò, F. and Faconti, G On the use of LOTOS to describe graphical interaction. In Proceedings of the HCI'92 Conference on People and Computers VII, Graphics - Design and Techniques, pages 155-173 (1992).

Duke, D. J. and Harrison, M. D. *Abstract interaction objects*. In Hubbold, R. J. and Juan, R., editors, Eurographics '93, pages 25-36, Oxford, UK. Eurographics, Blackwell Publishers (1993).

Roché, P.*, Modélisation et validation d'interface homme-machine*. Doctorat d'université, École Nationale Supérieure de l'Aéronautique et de l'Espace, (1998).

Markopoulos, P., Rowson, J., Johnson, P. *Composition and Synthesis with a Formal Interactor Model*, Interacting with Computers, 9, 197-223. (1997).

Markopoulos, P., *Interactors : formal architectural models of user interface software*. In Kent, A. and Williams, K.G, (Eds) Encyclopedia of Micocomputers, Volume 27 (Supplement 6), Marcel Dekker, New York, pp 203-235 (2001).

Olfa Daâssi, Gaëlle Calvary, Joëlle Coutaz, Alexandre Demeure. *Comet: a new generation of widget for supporting user interface plasticit*. IHM 2003 Proceedings of the 15th French-speaking conference on human-computer interaction on 15eme Conference Francophone sur l'Interaction Homme-Machine Pages 64-71(2003).

Gaëlle Calvary , Joëlle Coutaz , Olfa Dâassi , Lionel Balme , Alexandre Demeure, *Towards a new generation of widgets for supporting software plasticity: the "comet"* (2004).

Keling Da, Marc Dalmau, Philippe Roose – *Kalimucho : Middleware for Mobile Applications* – ACM SAC 2014 – pp. 413-419 – ACM Press – 24-28/03 – Gyeongju, Korea, (2014).

Pierre Dibon, Marc Dalmau, Philippe Roose – *Ubiquitous Widgets – Designing interactions architecture for adaptive mobile applications* – International workshop on Internet of Things – Ideas and Perspectives (IoTIP-13), In conjunction with IEEE DCOSS 2013, pp. 331-336 – DOI 10.1109/DCOSS.2013.32, May 21-23, 2013, Boston, USA (2013).

Daniel Cheung-Foo-Wo, Jean-Yves Tigli, Stephane Lavirotte, Michel Riveill - *Wcomp: a Multi-Design Approach for Prototyping Applications using Heterogeneous* Resources - IEEE International Workshop on Rapid System Prototyping'06, pp. 119-125, (2006).

V.Hourdin, J.Y. Tigli, S. Lavirotte, G. Rey, and M. Riveill. *SLCA, composite services for ubiquitous computing.* In Jason Yi-Bing Lin, Han-Chieh Chao, and Peter Han Joo Chong, editors, Mobility Conference, page 11. ACM, (2008).

Leap Motion - https://www.leapmotion.com/