Distributed User Interfaces in Space and Time

Jérémie Melchior
Université catholique de Louvain, Louvain School of Management
Louvain Interaction Laboratory, Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
Jeremie.melchior@uclouvain.be

ABSTRACT
Distributed User Interfaces (DUIs) have been imagined in order to support end users in carrying out interactive tasks that could be distributed in space (e.g., some subtasks are carried out in different locations) and time (e.g., some subtasks are carried out during different time intervals, depending on who is contributing to the task. Classical interactive applications involving a single-user, single-context user interface are rarely developed in a way that distributing parts or whole of the user interface is made effective and efficient. In order to facilitate the deployment of such distributed user interfaces, this thesis provides the following contributions: a series of models capturing the various aspects of a DUI based on new concepts (i.e. distribution scene and scenario), an engineering method for specifying DUIs based on these concepts, and a supporting toolkit providing the developers with distribution primitives.

Author Keywords
Distributed User Interfaces, Mobility, Ubiquity, Design.

General Terms
Design, Experimentation, Human Factors, Verification.

ACM Classification Keywords
C.2.4 [**Computer-Communication Networks**]: Distributed systems – *Distributed applications*. D2.2 [**Software Engineering**]: Design Tools and Techniques – *Modules and interfaces; user interfaces.* H5.2 [**Information interfaces and presentation**]: User Interfaces – *graphical user interfaces, user interface management system (UIMS).*

INTRODUCTION
A *Distributed User Interface* (DUI) is hereby defined as any application User Interface (UI) whose components can be distributed across different displays of different computing platforms that are used by different users, whether they are working at the same place (co-located) or not (remote collaboration) [1,2,7,9]. Consequently, DUIs allow for the UI to be spread out over a set of displays/devices/platforms taking advantage of each display/device/platform's unique properties instead of residing on a single display/device/platform [1] with the interaction capabilities that are constrained on this display/device/platform. People use one or several computing devices every day. In order to improve applications, researchers try to provide usable *user interfaces* (UIs) for this purpose. But applications only run on one single platform. Now, there are concepts such as distributed applications and *distributed user interfaces* (DUI) [2,5,6,8,9,10]. While the first has become popular, the second is only used by some groups of researchers and are not ready to a public use.

Motivations
The main motivations can be described with two small examples from [8]. "A user of a tabletop surface may wish to grab and use a keyboard from a nearby PC". In this first example, we have a multi-device system that we would like to organize in another way. We would like to use the keyboard device with the computer as well as the tabletop surface. To generalize this small example, we would like to be able to choose the way devices are logically connected at running time of the system. "Or an application running on a Smartphone might discover that its battery is about to expire, and look for another device onto which it can migrate while offering minimal interruption to its user". The second example shows an example of smart application. We would like to have **independence between user interface and logical part of an application** [8]. In the Smartphone example, we notice that there are a strong coupling between the user interface and the devices on which it runs. The place where the application is displayed should not be dependent from where the application runs. Users and applications might organize the user interface across several devices without other constraints than physical ones. Due to the lack of a single and complete description of what an application should be, there are a lot of different ways to create applications. Depending on what aims the application, it will be created using a toolkit, a framework, an API, a software development kit which may be more specific to the domain of the application [7]. The diversity of ways to create interactive applications leads to different ways to realize the same application. A consequence of this wide choice is that it is not easy to choose the one which is the most appropriate for the application. The choice can reduce the functionalities because some improvements exists but with different solutions. A major problem in computing science is to use the powerful varieties of operating systems, interaction mechanisms and form factors to create a large and powerful world that could be widespread like in the same room or interactive space [9]. The objective of all the operating systems is to be the most effective platform as possible but there are still a lot of features that still need to appear. Almost all the applications are local and the only interaction mechanisms for which they are written are the basic keyboard and mouse. Multiplatform applications are

very specifics and represent a few percent of all the applications. In order to get more powerful applications, there is a need for more development around multiplatform, multiuser and multimodal. Applications such as office automation and drawing application are developed for a single user and a single platform [7].

Starting concepts

The main concepts used in the thesis are:

- A *device* is a single physical unit such as a computer, a mobile phone or a complex interaction platform.
- A *user interface* is the set of graphical components of an application.
- A *user* is a concept of a human person interacting with the system.
- A *context of use* describes the environment and the material in which he is. A context of use *C* is composed by a platform *P*, a user *U* and an environment *E*.
- The *platform* is the software architecture (such as an operating system) provided to the user to interact with the device.

The *context of use* considered in the thesis may be more complex than one single platform, user and environment. Depending where and when the user is accomplishing the tasks, she is evolving in an *environment*. For example, he can be at work, at home or traveling. Even the easiest tasks can become difficult if the environment is not appropriate.

THESIS

Two dimensions: time and space. An important aspect of the distribution is the way users interact with the application. Users may be working at the same time in a competitive way or cooperating together to increase the effectiveness of the work. There can be different users working on the same application but at different time. Multiuser can be sequential or concurrent on a single computer or on several computers. While some users are working on the same computer, other users may interact with them from other computers wherever they are.

Concerns. we propose a description of the distribution domain, a toolkit for creating DUIs, a catalog of distribution primitives and concepts of distribution graphs and scenarios. It allows applications to be distributed across multiple devices, multiple screens and for multiple users. The concerns that this thesis try to address are [2,5,6,7,8,9]:

- *Concern #1. Development of distributed user interfaces:* the development of DUI is not supported by usual tools. Most of the time, developers have to manage the development in their own way. A lot of time is spent on the development of DUIs mostly the distributed aspects.
- *Concern #2. Support for distribution of user interfaces at running time:* existing DUIs are limited to predefined applications and domains of application which lead to little support for the various possibilities of distribution.
- *Concern #3. Support for multi-user collaboration:* multi-user applications are developed in different ways depending on the use and domain of application. The lack of a common base is slowing down the development.
- *Concern #4. Execution control in the distributed environment:* the control of the distribution is a real problem when managing DUI systems [4]. The limitations are high especially with a fixed level of granularity. Some systems can replicate windows while not being able to replicate widgets. Others can manipulate widgets one at a time but no group of widgets.
- *Concern #5. Network transparency:* The distribution of the UIs has to be network transparent in the sense that the user should not have to worry about network details such as IP address, user network and network settings.
- *Concern #6. Lack of description of the distributed domain and models:* The researches around multi-user applications and distributed user interfaces are very specifics to the needs of the developers and are almost never documented or badly documented.

Model-based Approach. The main contribution we bring to DUI is a model-based approach for designing distributed user interfaces (DUIs), i.e. graphical user interfaces that are distributed along one or many of the following dimensions: end user, display device, computing platform, and physical environment. The three pillars of this model-based approach are: (i) a Concrete User Interface model for DUIs incorporating the distribution dimensions and able to express in a XML-compliant format any DUI element until the granularity of an individual DUI element is reached, (ii) a specification language for DUI distribution primitives that have been defined in a user interface toolkit, and (iii), a step-wise method for modeling a DUI based several concepts we introduce in the thesis. The model-based approach for DUIs consists of conducting the following steps:

1. Build a cluster model of the platforms.
2. Build a CUI model for each platform.
3. Assemble models in the distribution scene.
4. Write a distribution scenario based on distribution primitives.
5. Develop the distribution scenario

Underlying models. The Concrete User Interface (CUI) model is independent of any computing platform and implementation language. A CUI model is hereby defined as recursive hierarchy of containers (e.g., windows, tabbed dialog boxes, group boxes) and individual widgets (e.g., check boxes, push buttons, list boxes, etc.). Widgets are laid out either horizontally or vertically. Each widget is defined as a vector $W=(P_i, V_i)$ where P_i denotes the ith property of the widget and V_i denotes the value of this property (e.g., the background color of a push button is grey). A selector consists of a selection of UI element types of a particular CUI model that satisfy a first-order predicate logical formula. In this way, a template is applied for a selector instead of a (potentially long) sequence of widgets as:

1. *Universal Selector*: applies the template to all UI elements belonging to a particular CUI, whatever they are.

2. *Element Type Selector*: applies the template to all UI elements belonging to a particular CUI which correspond to the selector's type (e.g., all containers).
3. *Class Selector*: applies the template to all UI elements belonging to a particular CUI.
4. *Identifier Selector*: applies the template to only one UI element belonging to a particular CUI: the one whose id property matches the string contained in the parameter.

We also introduce a platform model. The UI distribution concerns the repartition of one or many UI elements from one or many DUIs in order to support one or many users to carry out one or many tasks on one or many domains in one or many contexts of use, each context of use consisting of users, platforms, and environments. Therefore, the context of use is hereby considered as a cluster of individual components. In order to represent this cluster, we adopted the *Delivery Context Ontology* (DCO) standardized by W3C (www.w3.org/TR/dcontology), a subset of which in Fig. 1.

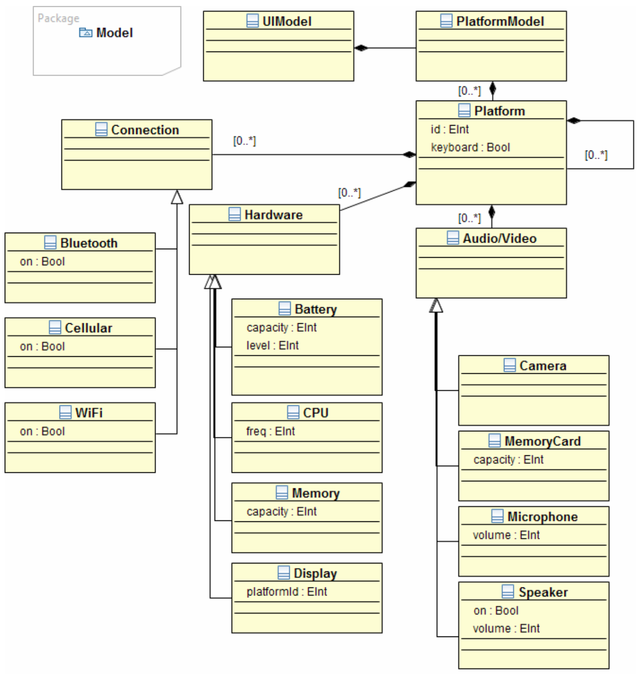


Figure 1. The platform model used for DUI.

According to DCO, a platform model is divided into one or many platforms. For example, a laptop itself consists of three platforms: the laptop, the display and the keyboard. Each platform has three main categories of components: a *connection* category representing the input and output connections to other devices or to the internet, the *hardware* category that defines the main components such as the CPU, the memory and if the platform has a display or not, and any component linked to the medias (e.g., audio and video). Based on this, a *cluster* is defined by a graph $G=(N_j,R_j)$ which is a set of nodes N_j connected together through R_j relationships. Each *node* consists of a DCO-compliant platform model representing any kind of device or components able to interact with the system (e.g., computers, displays, keyboards and mice are representative examples). Each relationship represents a communication channel (e.g., a Wi-Fi network or a Bluetooth connection) between nodes. Fig. 2a denotes a cluster composed of three platforms: a laptop connected to a flat monitor and a mobile phone. In order to properly express DUIs, to operate them and to reason on them, it is required to know at time what DUI is residing on which platform of the cluster. For this purpose, we hereby define a *distribution scene* as a cluster in which each node is associated to a CUI model, all CUI models connected to each other by a graph (Fig. 2b). Any cluster node contains a reference to a particular CUI model that could evolve over time. Consequently, a distribution scene holds a two-layer structure: (1) a cluster representing the physical setup of interaction elements and devices and (2) an associated graph of CUI models attached to any element in this cluster that supports some interaction. Not all platforms run a UI at any-time. To depict this, full circles in Fig. 2a represent that no DUI exist for those two platforms at some point (e.g., the starting time). The dashed circle around the laptop means that it holds a DUI. All models manipulated in this approach have their semantics defined in a UML V2.0 class diagram, a concrete syntax defined via a EBNF, and their stylistics defined.

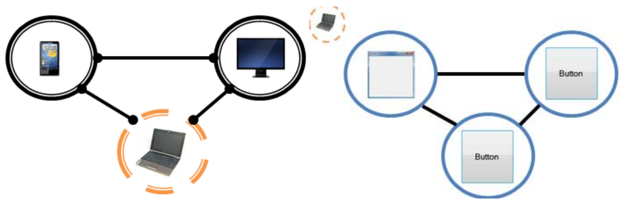


Figure 2. A distribution scene made up of: a cluster of three platforms (a) and an associated graph of CUI models (b).

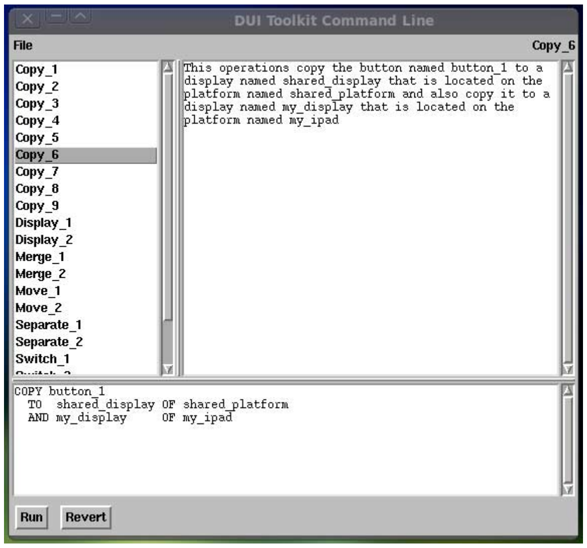
Catalog of distribution primitives. A distribution primitive consists of a basic operation in order to support distribution of any element of the CUI model of the cluster with respect to multiple devices/displays. The syntax of these distribution primitives is defined through an Extended Backus Naur Form (EBNF) grammar. Instances of distribution primitives are called by *statements*. The definitions of an operation, a source, a target, a selector and some other ones are defined in Fig. 3 (excerpt only). For instance, COPY button_1 TO button_2 ON shared_display means that button_1 is copied on shared_display and identify it as button_2. In this example, button_2 inherits everything from button_1, both presentation and behavior. This may induce some prioritization aspects since two push buttons could trigger the same function for instance. We are now working on extending these primitives in order to transfer partial/total presentation and/or behavior. For instance, button_2 could inherit the presentation from button-1, but its behavior will be expanded in order to address multi-user aspects. Or vice versa: button_2 could inherit the behavior of button_1, but its presentation will be changed. Different types of behavior inheritances are under study depending on which interaction status should be preserved.

```
statement = operation , white_space , source , white_space , ”TO” ,
white_space , target ;
operation = "SET" | "DISPLAY" | "UNDISPLAY" | "COPY" |
"MOVE" | "REPLACE" | "TRANSFORM" | "MERGE" |
"SWITCH" | "SEPARATE" | "DISTRIBUTE";
source = selector ;
target = displays | selector , white_space , “ON” ,
white_space , displays ;
displays = display_platform , { “,” , display_platform}
display_platform = display , [ white_space , “OF” ,
white_space , platform] ; selector = identifier , { “,” , identifier
} | universal ;
display = identifier ; platform = identifier ;
```

Figure 3. EBNF grammar for distribution primitives.

Toolkit for Distribution Primitives. A toolkit is being developed upon the aforementioned model-based approach in order to provide the developer with the distribution primitives of the catalog. It creates application with UI separated in two-parts: the proxy and the rendering. The proxy is represented as a separate part of the application than the rendering. The first keeps the state of the application and ensures the core functionalities, while the second displays the user interface. Application supporting DUI allows the rendering to be distributed on other platforms while the proxy stays where the application has been created. The toolkit works in an environment supported by Microsoft Windows XP and up, Apple Mac OS X, Linux and Android. We are currently working on Apple iOS. The applications created with this toolkit are multi-platform. Each graphical component is described as a record containing several keys and values. It ensures compatibility with XML because the keys/values become the name/value pairs of the XML.

Multiple meta-user interfaces. Any DUI based on the distribution primitives can be controlled by a meta-user interface [3] with the following interaction styles: command line interface, menu selection, drag and drop (partially), and programming language. Further investigation is needed in order to determine which interaction style is appropriate.



```
This operations copy the button named button_1 to a
display named shared_display that is located on the
platform named shared_platform and also copy it to a
display named my_display that is located on the
platform named my_ipad
```

```
COPY button_1
  TO  shared_display OF shared_platform
  AND my_display     OF my_ipad
```

Figure 4. Command line and menu selection for distribution.

CONCLUSION AND FUTURE WORK

The very first step in order to come up with a full method for designing DUIs consists of investigating which models, languages, development approach, and supporting software could be defined and implemented. This is why distribution primitives have been defined prior to any other step. But it is not because a distribution primitive could be implemented in a more effective way than by hand that the resulting DUI is usable for the end user. Multiple interaction styles exist that could support the same distribution primitive. In order to become effective, a DUI resulting from the aforementioned model-based approach should be offered via different interaction styles that are appropriate for the end user and task. So far, the metaphors used to control the DUIs are limited to keyboard/mouse interactions. For this purpose, we are conducting an experiment that would determine what are the end user preferences for some interaction style for a distribution primitive. We would like to extend these interactions by adding touch and multi-touch gestures to enable the distribution primitives. We think that the more natural the DUIs will be presented to the public, the better it will be for users.

REFERENCES

1. Beale, R. and Edmonson, W. Multiple Carets, Multiple Screens and Multi-Tasking: New Behaviours with Multiple Computers. In *Proc. of HCI'2007*, British Computer Society.
2. Berglund, E. and Bång, M. Requirements for distributed user interface in ubiquitous computing networks. *In Proc. of Conf. on Mobile and Ubiquitous MultiMedia MUM'2002*
3. Coutaz, J. Meta-User Interfaces for Ambient Spaces. In *Proc. of TAMODIA'2006*. LNCS, Vol. 4385. Springer, pp. 1–15.
4. Coyette, A., Vanderdonckt, J. A Sketching Tool for Designing Anyuser, Anyplatform, Anywhere User Interfaces. In: Proc. of 10th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2005 (Rome, 12-16 September 2005). LNCS, Vol. 3585, Springer-Verlag, Berlin, 2005, pp. 550-564.
5. Luyten, K. and Coninx, K. Distributed User Interface Elements to support Smart Interaction Spaces. In *Proc. of the 7th IEEE Int. Symposium on Multimedia*, (2005), pp. 277-286.
6. Luyten, K., Van den Bergh, J., Vandervelpen, Ch., and Coninx, K. Designing distributed user interfaces for ambient intelligent environments using models and simulations. *Computer & Graphics. 30*, 5 (2006) 702-713.
7. Melchior, J., Grolaux, D., Vanderdonckt, J., and Van Roy, P. A toolkit for peer-to-peer distributed user interfaces: concepts, implementation, and applications. In: *Proc. of EICS'09* (Pittsburgh, July 15-17, 2009). ACM Press, NY (2009), pp. 69–78.
8. Qiu, X.F. and Graham, T.C.N. Flexible and efficient platform modeling for distributed interactive systems. In: (Pittsburgh, July 15-17, 2009). ACM Press, New York (2009) pp. 29-34.
9. Vandervelpen, Ch., Vanderhulst, G., Luyten, K., and Coninx, K. Light-Weight Distributed Web Interfaces: Preparing the Web for Heterogeneous Environments. In *Proc. of ICWE'2005*. Springer-Verlag, Berlin (2005), pp. 197-202.
10. Vanderdonckt, J. Distributed User Interfaces: How to Distribute User Interface Elements across Users, Platforms, and Environments. In: *Proc. of XIth Congreso Internacional de Interacción Persona-Ordenador Interacción'2010* (Valencia, 7-10 September 2010). AIPO, Valencia (2010), pp. 3–14.