A Model-Based Approach for Distributed User Interfaces

Jérémie Melchior[1], **Jean Vanderdonckt**[1], **and Peter Van Roy**[2]
Université catholique de Louvain
[1]Louvain School of Management, Place des Doyens, 1 – B-1348 Louvain-la-Neuve, Belgium
[2]Computer Science Department, Place Sainte-Barbe, 2 – B-1348 Louvain-la-Neuve, Belgium
+32 10 47 {8379, 8525, 8374} - {jeremie.melchior, jean.vanderdonckt, peter.vanroy}@uclouvain.be

ABSTRACT
This paper describes 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 on the concepts of distribution graph expressing a distribution scenario that can be played namely based on the distribution primitives. A distribution graph consists of a state-transition diagram whose states represent significant distribution states of a DUI and whose transitions are labeled by an even-condition-action representation. The actions involved in this format may call any distribution primitive of the DUI toolkit. In order to exemplify this model-based approach, two simple DUIs are first designed: a DUI for the Pictionary game and a DUI for the Minesweeper game. They are then incorporated into a larger DUI game of the goose where cells may trigger the two other games.

Authors Keywords
Distribution graph, primitives, and scenario, Distributed User Interface, Meta-user interface, Shared displays, Ubiquitous computing, User Interface Toolkit.

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*. D2.m [**Software Engineering**]: Miscellaneous – Rapid Prototyping; reusable software. H5.2 [**Information interfaces and presentation**]: User Interfaces – *graphical user interfaces, user interface management system (UIMS)*.

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

INTRODUCTION
On the side of the *demand*, end users are more frequently involved in a context of use where domain objects are widespread, where roles and groups are configured in dynamic fashions, thus increasing the need for User Interfaces (UIs) that support them in these multiple configurations. On the side of the *offer*, the market has disseminated a large amount of computing platforms ranging from smartphones to wall screen displays, thus offering a wide spectrum of interaction surfaces to interact with. End users are however puzzled by what type of platform they should choose for a particular task, especially when several tasks are distributed in time and space. For instance, when an end user delegates a task to a colleague, parts or whole of this task UI should be transferred as well to the colleague. Even at run-time, an end user may want to ask for advice for a remote colleague, thus requesting to access to the currently running UI.

All these examples strive for a global approach for designing a *Distributed User Interface* (DUI), which is hereby defined as any application UI whose components can be distributed across varying displays of varying platforms that are used by varying users, whether they are working at the same place (co-located) or not (remote collaboration). DUIs have been successfully used in various domains of human activity (e.g., ambient intelligence [], clinical systems) and in computer science (e.g., migratory systems [], service-oriented architecture [], ubiquitous computing []). DUIs are fundamental and important because several applications require the integration of distributed interaction devices as functional wholes. There are two main categories in which they can be important. The user needs for DUI and addition to the limited UI development. Many work situations need collaboration between users, they share their computing tasks and so phithey should be able to share their UIs [?]. Current toolkits such as Java Swing, Microsoft Foundation Classes do not support DUI or any kind of distribution [?].

The remainder of this paper is structured as follows: the next section reports on some related work. Then, a specification language for UI distribution primitives is motivated, defined, and exemplified, based on a model of a Concrete UI. Then, a step-wise method for modeling a DUI based on a distribution scenario represented as a distribution graph involving the abode primitives is presented. A progressive case study will then exemplify how this method can be applied on games that are intrinsically challenging and distributed by nature, first

individually, then composed. Finally, a conclusion delivers the main points of this research and presents some future avenues.

RELATED WORK

How to support DUIs?

Berglund have set requirements that DUI have to meet: dynamic construction and distribution, make the best possible use of the available resources, provide graceful degradation of interaction, negotiate the interaction resources needs to be handled dynamically and transparent configuration and distribution of the UI. Tools supporting Distributed User Interface has to be developed [4-berglund]. The UI distribution granularity of existing toolkits and frameworks is always at high level. For example, in IMPROMPTU framework [6-bhiel], windows can be distributed across several devices. There are several possibilities for the granularity. The distribution can be at applications, windows, widgets and pixels levels. Almost every toolkit supporting distribution supports applications and/or windows levels. Some problems exist for the high-level granularity distributions. It raises security an d privacy problems. When allowing the distribution of a window or application, it allows all users to control or observe all shared windows. But sometimes, users would prefer to avoid users being able to get some windows or applications.

Compared to distribution programming which only focuses on the distribution of the core functionalities of an application, DUI's support is only at the beginning. To be able to develop application with fully controllable application, more research has to be done.

DUIs should be integrated in domains such as workflows, collaborative tools and for user that would like ubiquitous computing across several devices.

Consequently, DUIs allow for the UI to be spread out over a set of displays/devices/platforms taking advantage of their unique properties instead of residing on a single display/device/platform with the interaction capabilities that are constrained on this display/device/platform [??]. DUIs have been subject to several studies that investigate further their specific characteristics that may lead to design implications. This includes the use of multiple monitors on the same computing platform by a single user, the use of multiple platforms by a single user with synchronization between, exchange of information between platforms belonging to different users (e.g., by the Pick & Drop interaction technique), moving information between displays on a single platforms, partition of tasks across displays for a single user, sharing common information on a common display while keeping some information private on a own platform. Beale and Edmondson conducted user surveys in order to determine the user behavior induced by using a DUI: they identified the importance of having multiple carets and the complexity of multi-tasking and they suggest design implications for using DUIs in order to support distributed tasks. In particular, they stressed the importance of a multi-tasking model that is partially built at the local level of a single user and at the global level across users when collaboration exists. The global scenario should be also dissolved into local scenario in order to preserve the consistency between common tasks and individual tasks. This observation is fundamental for the work conducted here. Tan & Czewinsky found that physical discontinuities had no effect on performance, but found a detrimental effect from separating information within the visual field, when also separated by depth. Due to the multiplicity of interaction techniques in DUIs, Nacenta et al. conducted a study to compare the efficiency of six techniques for moving objects from a platform to another in four different distance ranges and with three movement directions. Their study suggests that spatial manipulation of data was faster than pressure-based techniques.

On the one hand, more user studies are available on specific DUI setups that provide us with more knowledge on design implications for such DUIs. Yet, in order to allow for the user to get the best potential of interaction capabilities offered by the various devices/displays/platforms for the current task to be carried out, we should enable designers as well as developers to provide users with the best DUI possible for a given set of devices/displays/platforms by described them in a formal way. This will allow both designers and developers to enable the underlying system to decide where different DUI portions should be placed in locations that are significant and usable for a distributed task to take place. For instance, the game of Pictionary is a typical example of a distributed task; one player selects a word from a dictionary, a second player draws this word on a surface shared by other players who have to guess what this word is as quick as possible, but below a certain time threshold. The team to which the winning player belongs to receives points.

Sjölund *et al.* implemented a DUI consisting of a remote control GUI on a smartphone that controls Windows Media Player displayed on a TV. Some controls of the Windows Media Player (e.g., play, stop, volume up, volume down) are moved to the smartphone and adapted to this platform at the same time. The

In Mightweight

As define in [4-berglund], such DUI components are treated as network-entities of self-configuring peer groups that negotiate UI responsibilities.

Usability of DUIs. Grudin [18-grudin] enlights usability issues with Multi-Display such as lack of support and mobile device not used to display.

Shortcomings of related work. In most of the aforementioned case studies, the distribution of UI elements is *predefined* and *opportunistic*. For example, in Sjölund, there is no other way to change the repartition of UI elements across the smartphone and the TV. In addition, it is not *replicable*. If another platform comes in, it is impossible to replicate or migrate on this platform the part that has already been

transferred to the first smartphone. In Lightweight, this is also the case, but it is more flexible in the distribution of services: once a service is selected, it can be distributed to any platform, existing or arriving, but a service can be distributed only once. On the one hand, this does not create any conflict, but on the other hand, it does not support *replication*.

Lack of design/development support

Limited granularity

No high level approach

Examples of applications able to distribute UI components without enough control. The key control is to strive for end user control of the distribution.

UI, DUI, MetaUI, context

CONTRIBUTIONS

The first contribution introduced is the distribution graph. It models some aspects of the distribution. Each node can be either any kind of device or components able to interact with the system or allowing the system to interact with. Computers, displays, keyboards and mice are some examples of possible nodes. They will be described by a platform model to know the main features of the device.

Figure 1: Four examples of devices

Or they can be applications. Distributed applications will be a graph with nodes being a part of the application running on a device.

Figure 2: Two examples of application

Or nodes can also be UI elements such as elementary widgets or complex widgets.

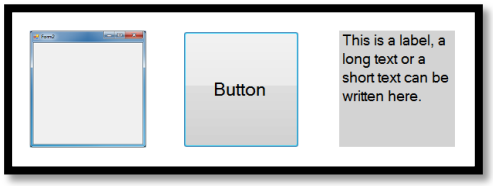

Figure 3: Three examples of graphical widgets

In order to have a clear view of the distribution, the distribution graph will be separated in a two-layer structure. The first, the global distribution graph, is the high-level view on the distribution space. The second layer is the local distribution graph.

For example, a global distribution graph of a common scenario is represented in the Figure 4. In this schema, we see a user having a Smartphone, a laptop and a display (either if it is connected to the phone or the laptop).

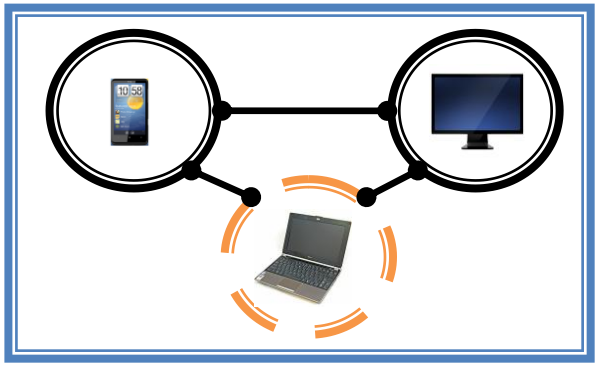

Figure 4: Example of global distribution graph

In the Figure, two nodes are full circle that represents the fact that no application and widget is in the node. The dashed circle around the laptop means that the laptop supports some applications and some widgets.

The second contribution is the platform meta-model. It allows developer to describe the different features of the devices through platform models. It is based on Delivery Context Ontology [10]. The platform model for this example is represented in Figure 5.

Devices/ Feature	***Mobile Phone***	***Laptop***	***Monitor***
Battery	YES : 100%	YES : 100%	NO
Bluetooth	YES : OFF	YES : OFF	NO
Camera	YES : 5MP	YES : 1MP	NO
Cellular	YES : HSDPA	NO	NO
CPU	YES : 1Ghz	YES : 2Ghz	NO
Display	YES : WVGA	YES : SXGA+	YES : HD 1080
Keyboard	YES	YES	NO
Memory Card	YES : 6Go	NO	NO
Microphone	YES	YES	NO
Speaker	YES	YES	NO
WiFi	YES : OFF	YES : ON	NO
Memory	512 MB	2 GB	NO

Figure 5: Platform model for the three devices of the example in Figure 4

To see more information about what is currently used by the laptop, we can see the local distribution graph of the laptop as in Figure 6.

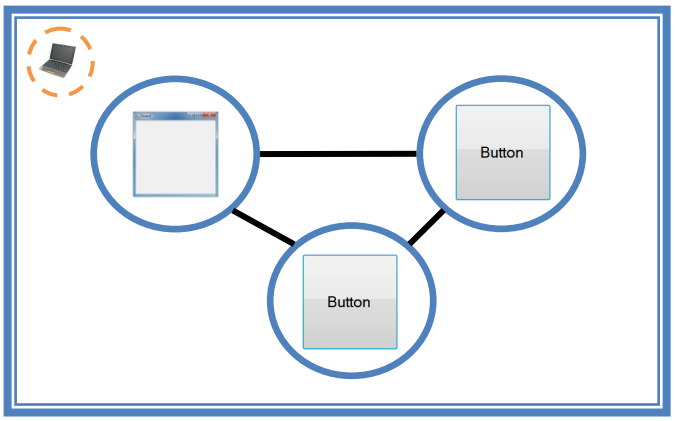

Figure 6: Local distribution graph of the laptop from Figure 4

A meta-UI is dedicated to control the distribution of some user interfaces components. An application needs at least one meta-UI. It allows the distribution of graphic components, widgets and container, from a platform to another. A model is introduced in order to represent the distribution mechanisms.

The model-based approach for designing DUI supports the distribution at several granularity levels. It starts at the widget level but also supports application and windows levels by distributing containers. The contribution here is the ability to merge UI from different applications, the ability to share parts of windows avoiding privacy problems and giving more control of the distribution to the users.

A third contribution is a distribution language to allow the distribution to be controlled by distribution statements. The language is very basic. A statement is defined as in Figure 7.

```
statement = operation , white_space , source , white_space , ”TO” , white_space , target ;
```

Figure 7: Definition of a statement in the distribution language

The definitions of an operation, a source, a target, a selector and some other ones are in Figure 8. The definitions could be extended later to support more operations or features.

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

Figure 8: Definition of main terms

The fourth contribution comes from these definitions; we allow developers and users to distribute their UI through distribution statements. For this, we first have to define the concept of selector.

A *selector* consists of a definition of the types of Common Information Model (CIM) elements to which the template applies, and a series of property declarations that define the template. Three major types of selector scope are considered:

- universalSelector: applies the template to all elements belonging to the CIM.
- elementTypeSelector: applies the template to all elements belonging to the CIM which correspond to the selector's type (e.g., all containers, all list boxes).
- classSelector: applies the template to all elements belonging to the CIM which correspond to the selector's type whose definition makes them part of the class (e.g., all containers having an id greater or equal to CC2, all list boxes having more than 10 items).
- idSelector: applies the template to only one element belonging to the CIM: the one whose id attribute matches the string contained in the parameter.

These selectors allow the user to specify the source and destination of the operation.

In Figure 9, you can see the execution of the operation: DISPLAY button(text:”B”).

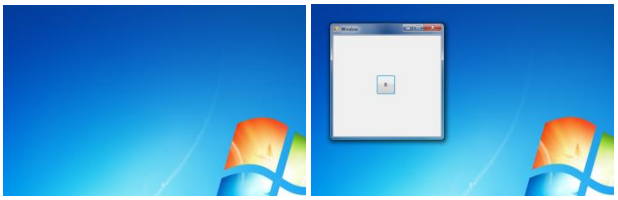

Figure 9: Display button(text:"B") operation

But operations are more complex than this example. For example, the operation COPY <Source > TO <Target> can either be:

1. COPY button_1 TO shared_display: simple copy of button_1 sent to shared_display without specifying neither an identifier nor a container
2. COPY button_1 TO button_2 ON shared_display: copy button_1 on shared_display and identify it as button_2
3. COPY button_1 TO button_2 ON shared_display of shared_platform: the same but we specify the shared_platform to avoid searching through all the platforms
4. COPY button_1, button_2 TO shared_display: copy button_1 and button_2 to shared_display in a single operation
5. COPY button_1 TO shared_display, my_display: copy button_1 to shared_display and also to my_display
6. COPY button_1 TO shared_display OF shared_platform AND my_display OF my_ipad: copy button_1 to both shared_display and my_display, specifying on which platform is each display

7. COPY * TO shared_display: copy all the graphical components from the current UI to shared_display
8. COPY ALL buttons TO shared_display: copy all buttons to shared_display
9. COPY individuals TO shared_display: copy any individual concrete user interface object to shared_display

The source UI associated to these examples can be found in Figure 10.

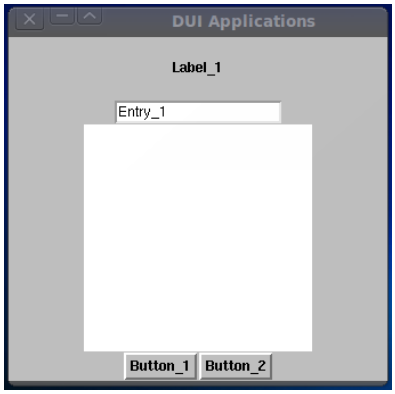

Figure 10: Example of Source UI for the COPY examples

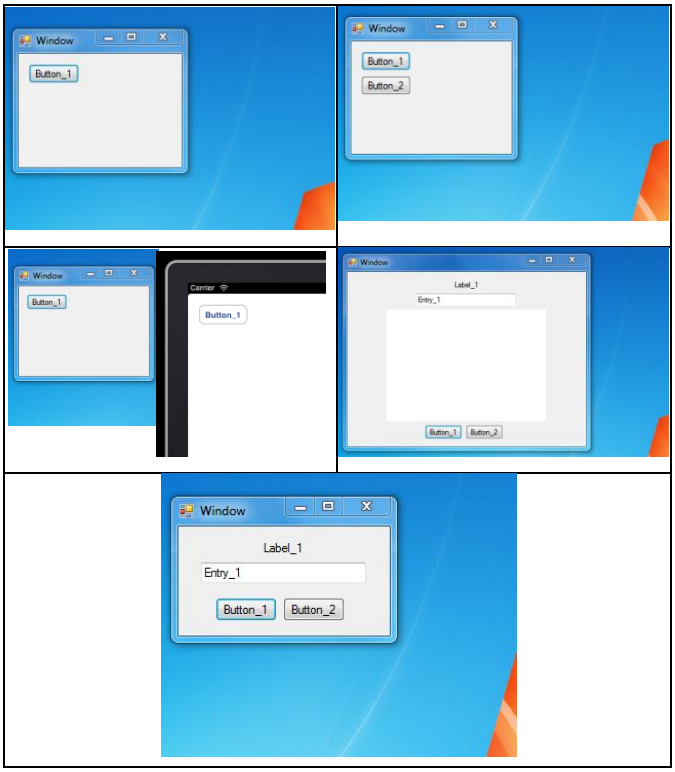

Figure 11: Result of examples 1, 2 & 3 (top left), 4, 8 (top right), 5 & 6 (middle left), 7 (middle right), 9 (bottom)

Assumption, distribution state, distribution primitives, distribution scenario, distribution graph, catalog of distribution operations

IMPLEMENTATION

A toolkit has been developed upon the model-based approach. It creates application with UI separated in two-parts: the proxy and the rendering. In Figure 12, 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.

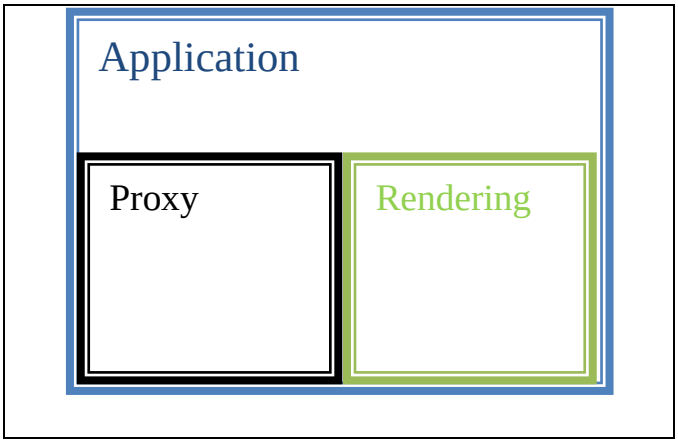

Figure 12: Structure of a DUI application

The toolkit works in an environment supported by Microsoft Windows operating systems (XP and newer), Apple Mac OS X, Linux and Android. And we are currently working on the full support for Apple iOS. The applications created with this toolkit are multi-platform.

We also based the toolkit on FormsVBT and a User Interface Description Language (UIDL).

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 markup.

The DUI can be controlled by a command line interface, a meta-UI or even by the applications themselves.

In Figure 13, there is the DUI command line interface which allows creating example and executing operations. It also works as a tutorial to understand how to use the DUI operations.

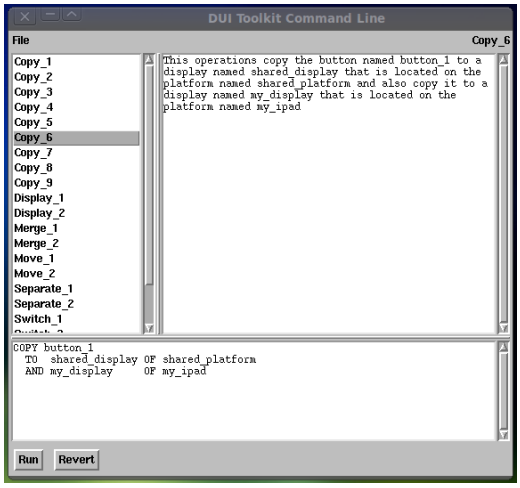

Figure 13: DUI Command Line Interface

EVALUATION AND CASE STUDY

A Pictionary game exemplified the mechanisms and evaluates the difference between a local Pictionary and a distributed Pictionary.

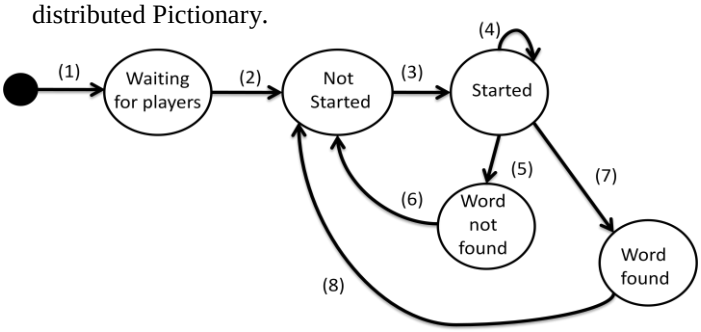

Figure 14: State-machine diagram of the Pictionary

In Figure 14, the STM shows that to start a game, the Pictionary needs several steps. The players will each have to assign or be assigned to a role. There are three roles: the player, the guessers and the observers. Two players are at least needed because the game needs one drawer and at least one guesser.

The distribution does not appear in the states of the Pictionary. The transition can be of two types: with and without distribution. As in any STM, the transitions may have some guarding conditions. For readability issue, we put it apart from the STM. Also, the final state is not display on the diagram. In order to clearly state the transitions, they are all numbered from 1 to 8 and here are the transitions in the form IF condition THEN action:

1. IF new_player_event
   THEN DISPLAY pictionary_UI TO new_player
2. IF nb_player > 1
   THEN DISPLAY assign_UI TO Pictionary_UI OF players
3. IF drawer != null && guesser != null
   THEN UNDISPLAY assign_UI
   DISPLAY draw_UI TO Pictionary_UI OF drawer
   AND guesser_UI TO Pictionary_UI OF guessers
   UPDATE observe_UI TO observers
4. UPDATE draw_UI
5. IF timer <= 0 & !found
   THEN UPDATE draw_UI, guesser_UI
6. UNDISPLAY draw_UI, guesser_UI, observer_UI
   DISPLAY assign_UI TO players
7. IF timer > 0 & found
   THEN UPDATE draw_UI, guesser_UI
8. UNDISPLAY draw_UI, guesser_UI,observer_UI
   DISPLAY assign_UI TO players

The drawer UI, in Figure 15, enables the drawing area. The guessers and observers are able to watch the drawing area but are unable to draw on it. The guessers can try words. A timer runs down until the word is found or until it reaches 00:00.

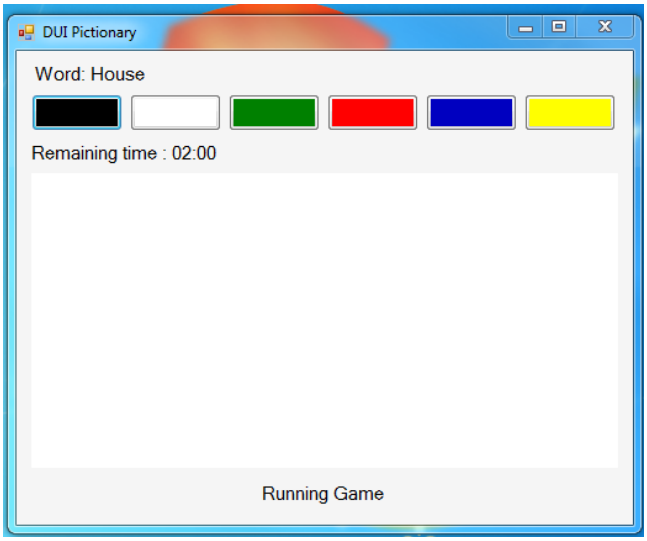

Figure 15: draw_UI enabling drawing area

This UI can be translated into a local distribution graph for the device on which the Pictionary is started. For example, a user with a computer and a mobile phone will have the following distribution graphs as in Figure 16 and 17.

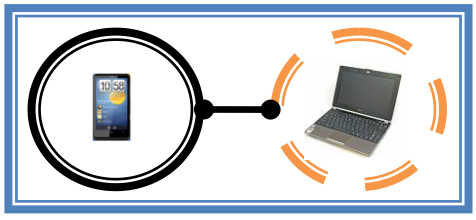

Figure 16: Example of a global distribution graph for the Pictionary

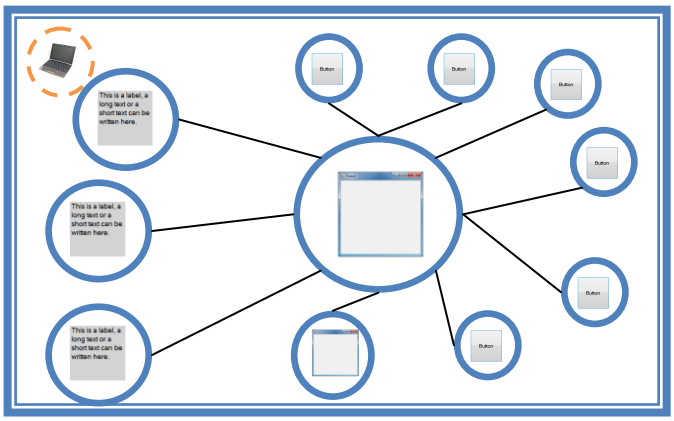

Figure 17: Example of a local distribution graph for the Pictionary

We also present an idea of a game that will be a combination of several games as the Game of the Goose.

A basic example would be to use the Pictionary, a Minesweeper and a Snake as games to combine. See Figure 18 for the Minesweeper examples of UI.

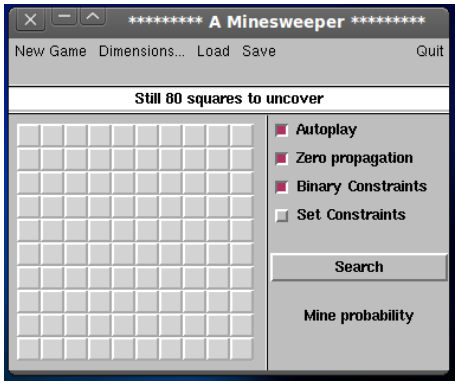

Figure 18: Example of a Minesweeper game

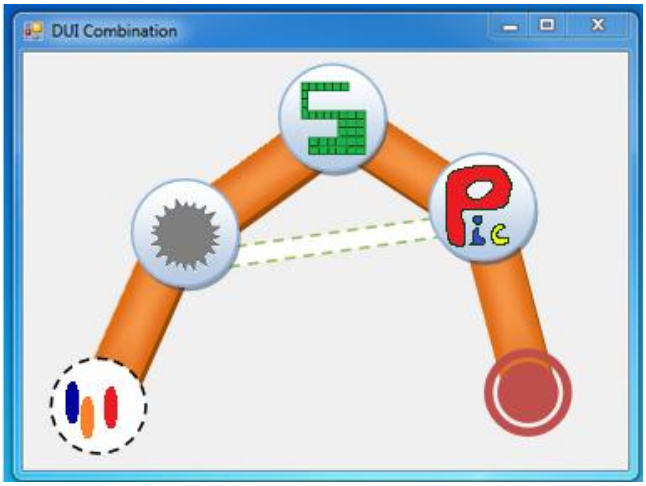

Figure 19: Example of UI for the combination game

The DUI combination game increases the use of distribution. When a player reaches a case on the board, the game of the case will be loaded and automatically display to its platforms. He may also redistribute the UI through its platforms at his own ease of use. There is a bridge between the Minesweeper and Pictionary games. A win in the Minesweeper game will allow the player to go to the Pictionary while a loss on the Pictionary will send the player back to the Minesweeper.

The example shown here is a simple 3-cases game, but the number should be higher for a real game. In the Game of the Goose, there are 63 cases.

CONCLUSIONS AND FUTURE WORK

In this paper we show how to design applications with Distributed User Interfaces enabling the control of the distribution of the UI components. We show that the support of multiple platforms is complex and needs adjustments. Operations supported by this model-based approach is not only basics such as display and hide components but as complex merging or dividing components. We have introduced concepts for better understanding the distribution. A language describes the statements that will allow the distribution of the UI. These statements are used in distribution scenarios for automatic distribution as well as in a command line interface to manually control the DUIs. Finally, the distribution graphs help modeling the distribution of the UI and describing the virtual distributed environment of the application.

No tool is developed in order to support the distribution graph representation. A toolkit supporting the creation of DUIs and the distribution operations is currently in development and will be introduced in the future.

REFERENCES

1. Ayatsuka, Y., Matsushita, N., and Rekimoto, J. HyperPalette: a hybrid computing environment for small computing devices. In: *Proc. of ACM Conf. on Human Factors in Computing Systems-Extended Abstracts CHI'00* (The Hague, April 1-6, 2000). ACM Press, New York (2000), pp. 133–134.
2. Bandelloni, R. and Paternò, F. Migratory user interfaces able to adapt to various interaction platforms. *Int. Journal of Human-Computer Studies 60*, 5-6 (2004), pp. 621–639.
3. Beale, R. and Edmondson, W. Multiple carets, multiple screens and multi-tasking: new behaviours with multiple computers. In: *Proc. of the 21st British HCI Group Annual Conf. on People and Computers HCI'07* (Lancaster, September 3-7, 2007). British Computer Society, Swinton (2007), pp. 55–64.
4. Berglund, E. and Bång, M. Requirements for distributed user interface in ubiquitous computing network. In: *Proc. of ACM Conf. on Mobile and Ubiquitous MultiMedia* MUM'02 (Oulu, December 11-13, 2002). ACM Press, New York (2002).
5. Bharat, K. A. and Cardelli, L. Migratory applications. In: *Proc. of the 8th ACM Symposium on User interface and Software Technology UIST'95* (Pittsburgh, November 15-17, 1995). ACM Press, New York, 1995, pp. 132–142.
6. Biehl, J. T., Baker, W. T., Bailey, B. P., Tan, D. S., Inkpen, K. M., and Czerwinski, M. IMPROMPTU: a new interaction framework for supporting collaboration in multiple display environments and its field evaluation for co-located software development. In: *Proc. of the 26th ACM Conf. on Human*

Factors in Computing Systems CHI'08 (Florence, April 5-10, 2008). ACM Press, New York (2008), pp. 939–948.
7. Chung, G. and Dewan, P. Towards dynamic collaboration architectures. In: *Proc. of the ACM Conf. on Computer Supported Cooperative Work CSCW'04* (Chicago, November 6-10, 2004). ACM Press, New York (2004), pp. 1–10.
8. Coutaz, J. Meta-User Interfaces for Ambient Spaces. In: *Proc. of 5th Int. Workshop on Task models and Diagrams for User Interface Design TAMODIA'2006* (Hasselt, October 23-24, 2006). LNCS, Vol. 4385. Springer, Berlin (2006), pp. 1–15.
9. Delivery Context Ontology (DCO), W3C, Geneva, 2010. http://www.w3.org/TR/2009/WD-dcontology-20090616/.
10. Demeure, A., Sottet, J.S., Calvary, G., Coutaz, J., Ganneau, V., and Vanderdonckt, J. The 4C Reference Model for Distributed User Interfaces. In: Proc. of 4th Int. Conf. on Autonomic and Autonomous Systems ICAS'2008 (Gosier, March 16-21, 2008), IEEE Comp. Soc. Press (2008), pp. 61-69.
11. Dewan, P. and Shen, H. Controlling access in multiuser interfaces. *ACM Trans. Comp.-Hum. Interact. 5, 1* (1998), 34-62.
12. Distributed Programming in Mozart–A Tutorial Introduction, chapter 3: Basic Operations and Examples, accessible at *http://www.mozart-oz.org/documentation/dstutorial/node3*. html#chapter.examples
13. Econometric Modeling & Computing Corporation (EMCC). 2010. Accessible at: http://www.speakeasy.com
14. Eisenstein, J., Vanderdonckt, J., and Puerta, A. Applying model-based techniques to the development of UIs for mobile computers. In: *Proc. of the 6th Int. Conf. on Intelligent User Interfaces IUI'01* (Santa Fe, January 14-17, 2001). ACM Press, New York (2001), pp. 69–76.
15. Grolaux, D., Van Roy, P., and Vanderdonckt, J. Migratable User Interfaces: Beyond Migratory User Interfaces. In: *Proc. of 1st IEEE-ACM Annual Int. Conf. on Mobile and Ubiquitous Systems: Networking and Services MOBIQUITOUS'04* (Cambridge, August 22-26, 2004). ACM Press (2004), pp. 422–430.
16. Grolaux, D., Vanderdonckt, J., and Van Roy, P. Attach me, Detach me, Assemble me like You Work. In: *Proc. of the 10th IFIP TC13 Int. Conf. on Human-Computer Interaction INTERACT'05* (Rome, September 12-16, 2005). LNCS, Vol. 3585, Springer-Verlag, Berlin (2005), pp. 198–212.
17. Grudin, J. Partitioning digital worlds: focal and peripheral awareness in multiple monitor use. In: *Proc. of the ACM Conf. on Human Factors in Computing Systems CHI'01* (Seattle, 2001). ACM Press, New York (2001), pp. 458–465.
18. Han, R., Perret, V., and Naghshineh, M. WebSplitter: a unified XML framework for multi-device collaborative Web browsing. In: *Proc. of the ACM Conf. on Computer Supported Cooperative Work CSCW'00* (Philadelphia, December 2-6, 2000). ACM Press, New York (2000), pp. 221–230.
19. Hutchings, D. R., Smith, G., Meyers, B., Czerwinski, M., and Robertson, G. Display space usage and window management operation comparisons between single monitor and multiple monitor users. In: *Proc. of the Working Conference on Advanced Visual Interfaces AVI'04* (Gallipoli, May 25-28, 2004). ACM Press, New York (2004), pp. 32–39.
20. Hutchings, H.M. and Pierce, J. S. Understanding the whethers, hows, and whys of divisible interfaces. In: *Proc. of the Working Conf. on Advanced Visual Interfaces AVI'06* (Venezia, May 23-26, 2006). ACM Press, New Y. (2006), pp. 274–277.
21. Loeser, C., Mueller, W., Berger, F., and Eikerling, H.J. Peer-to-Peer Networks for Virtual Home Environments. In: *Proc. of the 36th Annual Hawaii international Conference on System Sciences HICSS'03* (Big Island, January 6-9, 2003). IEEE Computer Society, Los Alamitos (2003), p. 282.
22. Lorenz, A. Research directions for the application of MVC in ambient computing environments. In: *Proc. of the 1st Int. Workshop on Pattern-Driven Engineering of interactive Computing Systems PEICS'10* (Berlin, July 20, 2010). ACM Press, New York (2010), pp. 28–31.
23. Luyten, K. and Coninx, K. Distributed User Interface Elements to support Smart Interaction Spaces. In: *Proc. of the 7th IEEE Int. Symposium on Multimedia ISM'2005* (December 12-14, 2005). IEEE Computer Society (2005), pp. 277–286.
24. Luyten, K., Vandervelpen, C., and Coninx, K. Migratable User Interface Descriptions in Component-Based Development. In: *Proc. of the 9th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2002* (Rostock, June 12-14, 2002). Lecture Notes in Computer Science, Vol. 2545. Springer-Verlag, London (2002), pp. 44–58.
25. Luyten, K., Van den Bergh, J., Vandervelpen, Ch., and Coninx, K. Designing distributed user interfaces for ambient intelligent environments using models and simulations. *Computers & Graphics 30*, 5 (2006), pp. 702–713.
26. 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 the 1st ACM Symposium on Engineering interactive Computing Systems EICS'09* (Pittsburgh, July 15-17, 2009). ACM, pp. 69–78.
27. Molina, J.P., Vanderdonckt, J., González, P., Fernández-Caballero, A., and Lozano, M.D. Rapid Prototyping of Distributed User Interfaces. In: Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2006 (Bucharest, 6-8 June 2006). Springer-Verlag, Berlin (2006), pp. 151–166.
28. Myers, B. A. 2001. Using handhelds and PCs together. *Commun. ACM 44*, 11 (November 2001), pp. 34–41.
29. Newman, M. W., Izadi, S., Edwards, W. K., Sedivy, J. Z., and Smith, T.F. User interfaces when and where they are needed: an infrastructure for recombinant computing. In: *Proc. of the 15th ACM Symposium on User interface Software and Technology UIST'02* (Paris, October 27-30, 2002). ACM Press, New York (2002), pp. 171–180.
30. Qiu, X. F. and Graham, T.N. Flexible and efficient platform modeling for distributed interactive systems. In: *Proc. of the 1st ACM Symposium on Engineering interactive Computing Systems EICS'09* (Pittsburgh, July 15 - 17, 2009). ACM Press, New York (2009), pp. 29–34.
31. Sjölund, M., Larsson, A., and Berglund, E. Smartphone Views: Building Multi-Device Distributed User Interfaces In: *Proc. of MobileHCI'2004* (Glasgow, 13-16 September 2004). LNCS, Vol. 3160. Springer, Berlin (2004), pp. 507–511.
32. Tan, D.S. and Czerwinski, M. Effects of Visual Separation and Physical Discontinuities when Distributing Information across Multiple Displays. In: *Proc. of IFIP TC13 Int. Conf. on Human-Computer Interaction INTERACT'03* (Zurich, September 1-5, 2003). IOS Press (2003), pp. 252–260.
33. Tan, D.S., Myers, B. and Czerwinski, M. 2004 WinCuts: Manipulating Arbitrary Window Regions for More Effective User of Screen Space. In: *Proc. of ACM Conf. on Human Factors in Computing Systems CHI'2004* (Vienna, April 24-29, 2004). ACM Press, New York (2004), pp. 1525–1528.

34. 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.
35. Vandervelpen, Ch., Vanderhulst, G., Luyten, K., and Coninx, K. 2005. Light-Weight Distributed Web Interfaces: Preparing the Web for Heterogeneous Environments. In: *Proc. of the 5th Int. Conf. on Web Engineering ICWE'2005* (Sydney, July 27-29, 2005). Springer, Berlin (2005), pp. 197–202.
36. Roy, P. V. and Haridi, S. 2004 Concepts, Techniques, and Models of Computer Programming. *MIT Press*.
37. Xiaojun, B. and Balakrishnan, R. Comparing usage of a large high-resolution display to single or dual desktop displays for daily work. In: *Proc. of the 27th Int. Conf. on Human factors in Computing Systems CHI'09* (Boston, April 4-9, 2009). ACM Press, New York (2004), pp. 1005–1014.
38. Yanagida, T. and Nonaka, H. 2008. Architecture for Migratory Adaptive User Interfaces. In: *Proc. of the 8th IEEE Int. Conf. on Computer and Information Technology CIT'2008* (Sidney, July 8-11, 2008). IEEE (2008), pp. 450–455.