Chapter 12

RAPID PROTOTYING OF DISTRIBUTED USER INTERFACES

José Pascual Molina Massó[1,2], Jean Vanderdonckt[1],Pascual González López[2], Antonio Fernández-Caballero[2], and María Dolores Lozano Pérez[2]
[1] School of Management (IAG), Université catholique de Louvain
Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
E-mail: {molina, vanderdonckt}@isys.ucl.ac.be
Tel: +32 10/478525 – Fax: +32 10/478324 – Web: http://www.isys.ucl.ac.be/bchi
[2] Lab. of User Interaction & Software Engineering
Inst. de Investigación en Informática de Albacete (I3A), Universidad de Castilla-La Mancha
Campus universitario s/n – S-02071 Albacete (Spain)
E-mail: {jpmolina, pgonzalez, caballer, mlozano}@dsi.uclm.es
Tel: +34 967/599200 – Fax: +34 967/599224 – Web: http://www.i3a.uclm.es

Abstract This paper introduces a software tool for rapid prototyping of interactive systems whose user interfaces could be distributed according to four axes defined in a design space: type of computing platform, amount of interaction surfaces, type of interaction surface, and type of user interface. This software is based on a virtual toolkit for rendering the user interfaces in a virtual world depicting the real world in which the distribution occurs. The virtual toolkit consists of a layer for rendering a concrete user interface specified in a user interface description language. This paper presents its extension to modeling the external environment in terms of the design space so as to render the context of use in which the user interfaces are distributed. For each axis, a pair of functions enables exploring the axis in decreasing and increasing order so as to explore various situations of distribution, axis by axis, or in a combined way. As the interfaces resulting from this rendering are truly executable ones, this system provides designers with an acceptable means for generating ideas about how a user interface can be distributed in a context of use, and helps to evaluate the quality of a solution at an early design stage. Four representative situations located on the design space are implemented and discussed: distribution in a multi-platform context, distribution of the workplace, ubiquitous computing, and ambient intelligence, thus proving the coverage of the design space and the capabilities of the whole system.

Keywords: Ambient intelligence, Context-aware computing, Distributed computing, Distributed interfaces, Pervasive computing, Ubiquitous computing, User interface extensible markup language, Workplace distribution.

1. INTRODUCTION

Distributed User Interfaces (DUIs) [1,6,14] apply the notion of distributing parts or whole of a user interface (UI) across several places or locations like Distributed Systems [12] do for general software. In studying DUIs, we have identified two main classes of problems: an ontological confusion about the various concepts and associated definitions expressing how and according to what to distribute a given UI; a practical problem of experimenting a DUI at early design time before developing it completely.

The first class of problems is motivated by observing that the several recent advances in DUIs (among them are [4,7,9,12,14,17,18,19,20]) do not necessarily rely on the same concepts of distribution and, when it is the case, the definition and/or the axes according to which the distribution may take place largely vary from one research to another. Although significant efforts exist to shed some light in this area and to structure DUI design issues, mainly in [1,6], the relationship between these design issues and their corresponding physical configurations are not always straightforward to establish.

The second class of problems poses even more challenges because developing DUIs eminently require a sophisticated architecture, and due to that level of sophistication it not surprising that DUIs are slow to obtain, expensive to produce, and probably equally complex to use.

The aforementioned observations show that designing a DUI remains a complex problem which may prevent designers from exploring design issues because of their associated cost. If the development cost of several DUIs is too high with respect to the benefit of exploring different design issues and physical configurations for distribution, it is likely that this exploration will be abandoned soon due to lack of flexibility. In addition, the usability issues raised by distributing a UI across one or several dimensions [10,21] are serious and could be hard to uncover before a really usable solution is found.

Therefore, we argue that rapid prototyping of DUIs [10,19] turns to be an important issue: not only rapid prototyping could be used as a vehicle for developing and demonstrating visions of innovative DUIs, but also they could help showing various distribution configurations before going to full implementation [2]. However, rapid prototyping is also a challenging problem, as the design space of DUIs covers a wide range of different possibilities. In order to tackle this complexity, the approach presented in this paper is to develop a software tool which supports rapid prototyping of DUIs based on a limited, but significant, set of four design dimensions.

The remainder of this paper is sequenced as follows: the four design dimensions for DUIs will be defined in a design space in Section 2, namely based on an environment model. This design space will be then exploited to compare related work in consistent terms in Section 3. Section 4 will exem-

plify four frequently found situations where UIs are distributed across one dimension of the design space considered at a time, while showing that combination is allowed. Each subsection is devoted to each design dimension. Section 5 will sum up the benefits of the rapid prototyping approach.

2. DESIGN SPACE FOR DISTRIBUTED USER INTERFACES

The design space described in this paper relies on an environmental model and four axes or dimensions which are explained in terms such as "digitization" or "dematerialization", to cite just a couple of them. But, prior to defining these concepts on which the rest of this paper will rely, there are others that also support this work and so they must be explained before continuing. These are the UI development framework used, and the notion of context of use and interaction surface.

2.1 Foundation

In this paper, it is assumed that the development of user interfaces relies on the CAMELEON framework [3], which structures the development life cycle of multi-target UIs according to four layers (Fig. 1):

1. The *Final UI* (FUI) is the operational UI, i.e. any UI running on a particular computing platform either by interpretation (e.g., through a Web browser) or by execution (e.g., after the compilation of code in an interactive development environment).
2. The *Concrete UI* (CUI) expresses any FUI independently of any term related to a peculiar rendering engine, that is, independently of any markup or programming language.
3. The *Abstract UI* (UI) expresses any CUI independently of any interaction modality (e.g., graphical, vocal, or tactile).
4. The *Task & Concept* level, which describes the various interactive tasks to be carried out by the end user and the manipulated domain objects.

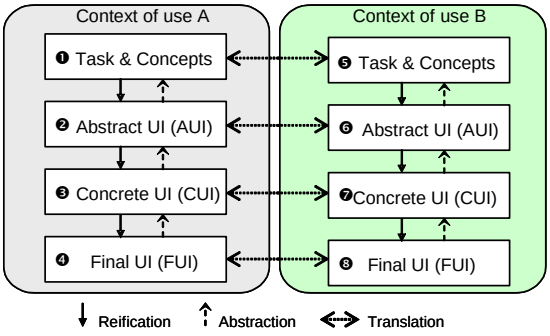

Figure 1. The Cameleon User Interface Reference Framework.

We refer to [3] for more details and to [13] for its translation into models uniformly expressed in the same User Interface Description Language (UIDL). The selected language for this work is UsiXML [13], which stands for User Interface eXtensible Markup Language (http://www.usixml.org). Any other UIDL could be used equally provided that the used concepts are also supported.

The *Context of use* describes all the entities that may influence how the user's task is carrying out with the future UI. It takes into account three relevant aspects, each aspect having its own associated attributes contained in a separate model: *user type* (e.g., system experience, task experience, task motivation), *computing platform* type (e.g., mobile platform vs. stationary one), and *physical environment* type (e.g., office conditions, outdoor conditions).

Finally, we define the concept of *interaction surface*, which was introduced by [5], as any physical surface which can be "acted on or observed" so as to support user interaction with a system, whether visible or embedded. For instance, an interaction surface could be a screen, a monitor, a wall display, a table equipped with camera tracking techniques, or a pad with projection. See [5] for a complete definition of physical (e.g., weight, size, material, shape, solidity/fluidity/nebulosity) and modality attributes.

2.2 Environment Model

For the purpose of this work, a richer environment model has expanded UsiXML's existing physical environment model with the concept of interaction surface. The physical environment (Fig. 2) is expanded with a characterization of its physical space, described as a scene which is in turn decomposed in surfaces, to be connected together or not through position and orientation. This characterization is deeply inspired by world modeling which is traditional to VRML97/X3D 3D worlds [16]. Each surface composing the physical space (e.g., the walls, the table, or the doors of a room) could be declared as an interaction surface which can be acted on or observed, depending on input/output.

A second addition is that each environment could comprise one or several computing platforms, each of them being characterized with a series of attributes. Each computing platform could be located precisely with respect to an environment surface and could hold none, one or many hardware platforms, which are declared as a general form of output (e.g., a display, a monitor, a screen). Each such platform is of course an interaction surface which could be acted on (by using pointers) and/or observed (by looking at the screen). Each interaction surface is defined by its shape, which is the area sensible to interaction. Hardware platforms are therefore considered as rectangular-shaped interaction surfaces. One could imagine probably other shapes like a circle or an oval, but in our implementation, only convex surfaces are subject to the FUI rendering.

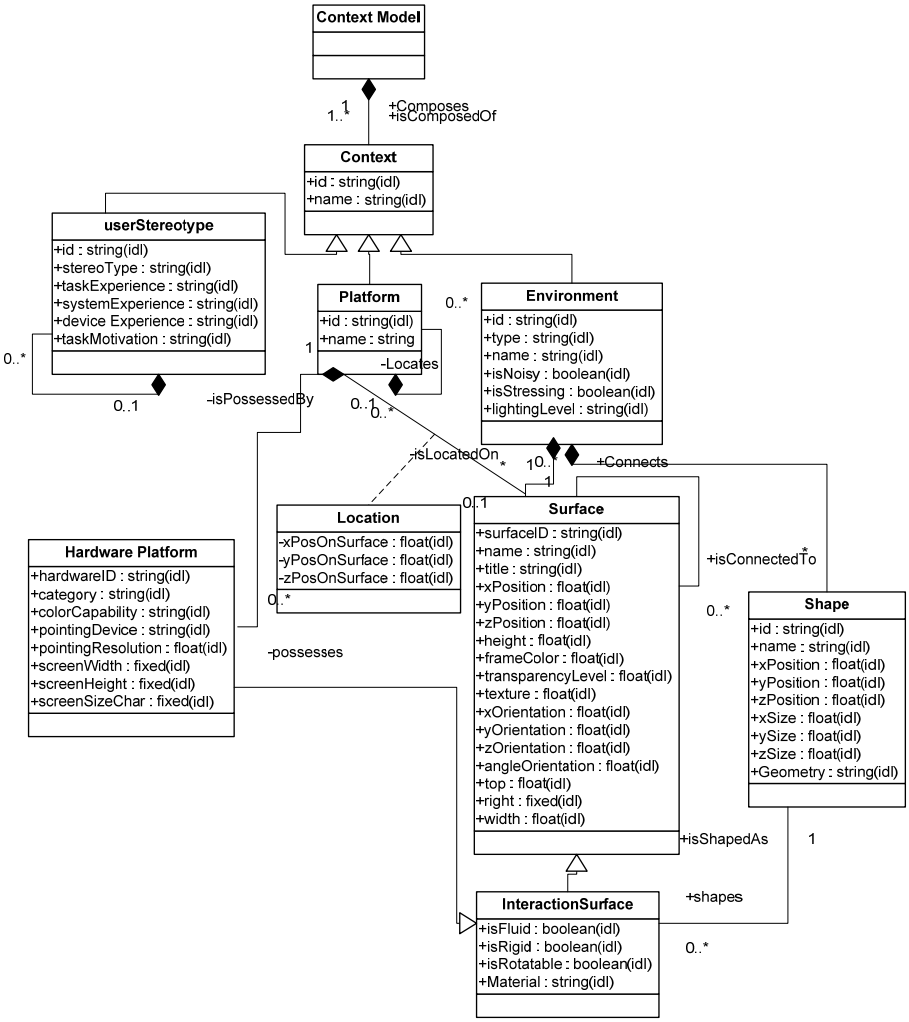

Figure 2. The expanded environment model, based on the notion of interaction surface [5].

2.3 Definition of the Design Space

The proposed DUI design space is decomposed into four design axes or dimensions (Fig. 3): type of computing platform, amount of interaction surfaces, type of interaction surface, and type of user interface. They are respectively explained in more detail in the following subsections. Lyytinen and Yoo [15] have studied how conventional systems may evolve in the future towards ubiquitous computing through two dimensions: the level of system integration within the ambient environment and the degree of mobility. These two dimensions correspond to type of interaction surface and type of computing platform respectively in the design space of Fig. 3.

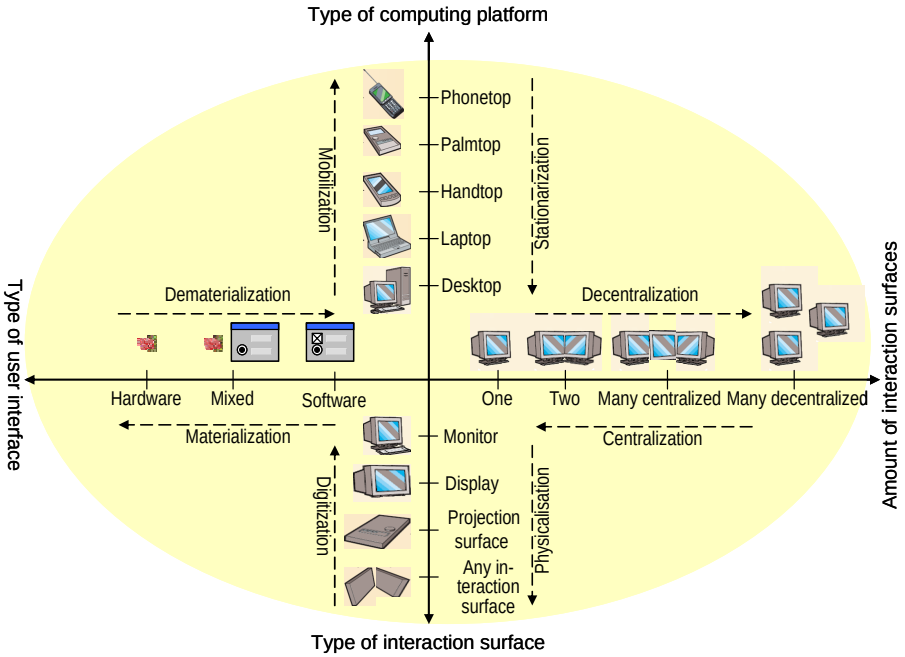

Figure 3. The four design dimensions of the DUI design space.

2.3.1 Type of Computing Platform

The *type of computing platform* represents the first axis along which a UI may be distributed, meaning here that parts or whole of an existing UI may transit from one computing platform to another. On this axis, the computing platforms are ranked by decreasing order of mobility:

- "Stationarization" is the process consisting of rendering a FUI on a target platform which is more stationary than the source platform. Each progressive graduation on this axis could be achieved by performing an abstraction from CUI to AUI, followed by reification from AUI to CUI, and then restricted by a selection to more stationary platforms [3]. A simple translation from CUI to CUI may also work.
- "Mobilization" is the inverse process. To support moving along this axis, various operations are diversely supported in the literature such as direct transfer (simple translation without any modification), migration, copying, or duplication [1].

2.3.2 Amount of Interaction Surfaces

The *amount of interaction surfaces* denotes how many interaction surfaces are used to render the DUI. Typical cases are: a single monitor per computing platform, two dual monitors [10], three to five monitor display wall (e.g., www.panoramtech.com, www.go-l.com), and many displays

which could be centralized per computing platform or decentralized, thus posing the problem of multiple foci of interest for one task [21]. These are the terms that characterize this axis:

- "Decentralization" is the process consisting of rendering a FUI on more interaction surfaces than previously, the displays being connected to the same computing platform or not. Each progressive graduation on this axis could be achieved by performing a decomposition of the source UI (e.g., by graceful degradation [8], by fragmentation [20], by semantic redesign [18]) which are then rendered on the various interaction surfaces.
- "Centralization" is the inverse process. To support it, various operations like union, merging, re-composition may occur before re-rendering the gathered pieces on less interaction surfaces.

2.3.3 Type of Interaction Surface

The *type of interaction surface* depicts the level of physicality of the interaction surface used to render the UI. A computer monitor or a public display are considered as *digital* interaction surfaces, as opposed to projection surfaces which are considered as *physical* ones since the UI is rendered by projection from a projector and user's events are tracked by camera recognition techniques borrowed from signal processing domain. Consequently:

- "Physicalization" is the process of rendering a FUI on a target interaction surface that is more physical than the source one. For this purpose, different rendering functions should be implemented depending on the type of interaction surface used. Whether the surface resolution changes, repurposing functions could be called to reshuffle the UI components.
- "Digitization" is the inverse process (to digitize = to convert data such as an image to a digital form – source: Merriam Webster On-line).

2.3.4 Type of User Interface

The *type of user interface* expresses the physicality level of the FUI components. A DUI is said to be *software* when all its components are traditional software widgets, *hardware* when all its components are physical objects like switches, buttons, and *mixed* when components could either be software or hardware simultaneously. Therefore:

- "Materialization" is the process consisting of changing the distribution of the FUI towards more physical components (to materialize = to cause to appear in bodily form – source: Merriam Webster On-line). Each progressive graduation on this axis could be achieved by performing a decomposition of a source FUI into smaller pieces and re-assigning dedicated pieces to physical objects instead of software objects. This obviously touches the area of tangible UIs.

- "Dematerialization" is the inverse process (to dematerialize = to cause to become or appear immaterial – source: Merriam Webster On-line). To support it, a FUI is decomposed into fragments, some of them being re-assigned to digital objects (e.g., widgets). When a digital object is transformed to a physical one, its CUI definition is abstracted [3] into an AUI counterpart, followed by a reification from AUI to CUI, and then restricted by a selection to only those objects belonging to the physical world. In this way, it is possible to find out an object in the physical world with an equivalent behavior, the presentation of which does not matter [22].

2.4 Software Tool for Rapid Prototyping

To support the various operations involved in the design space defined in Fig. 3, VUITOOLKIT [17] has been developed above UsiXML and expanded with the environment model of Fig. 2 so as to render a CUI as a FUI in a virtual world. First, the environment model gives rise to a virtual world composed on surfaces, some of them being interactive. In particular, computing platforms could be located on some of these surfaces or considered as an interaction surface per se. Second, the toolkit abstracts objects from Web3D languages (e.g., VRML, VRML97, X3D which are typically used in modeling virtual reality worlds and scenes). To bridge the gap between a UsiXML specification and its counterpart in virtual reality, and to render it properly in the virtual world, several cases could occur (Table 1):

1. *Direct mapping between a CUI and a Web3D primitive*. This mapping could be one-to-one (bijection) or one-to-many (composition of objects). It is not always possible to set a one-to-one mapping as those Web3D languages define basic elements such as shapes and sensors that must be used together to create interactive elements, e.g., 3D widgets. The new standard X3D does not change this status, even though it includes new 2D geometry nodes that make easier to draw 2D interfaces in a 3D world. Therefore, some basic widgets (e.g., a window) were redeveloped from scratch by assembling shapes together with their behavior.
2. *New mapping between a CUI and a Web3D counterpart*. Sometimes, no object exists natively in the Web3D language to ensure the mapping. In this case, there is a need to fill this gap by introducing a new widget in the Web3D world by appropriate implementation. This is what happens with the CUI used as a starting point for the toolkit, for each of them there is an element in the toolkit that can be used for their representation in the FUI. For this purpose, all widget classes defined in a UsiXML-compliant CUI have been sub-classed into their equivalent objects in virtual reality. Typical examples of these objects include check box, radio button, list box,

combo box, slider, and cursor. This correspondence remains incomplete: some CUI attributes are not subject to any rendering in the virtual world, some other attributes missing in the CUI definition are added because they are necessary to render their presentation and their behavior [16,17], such as those properties for the position and the dimensions of the widget.

3. *No possible mapping*. Despite the efforts to establish a mapping between any CUI and its counterpart, the UsiXML concept of box (the layout is decomposed into a hierarchy of vertical and horizontal boxes) is transformed into a system where coordinates of objects are computed from constraints imposed on them, instead of solving the constraints at rendering time. In case of rendering on a surface which is not a computing platform screen (e.g., a wall), surface attributes are exploited.

Table 1. Possible mappings between 2D and 3D.

CUI level (UsiXML)	VUIToolkit (VRML97 / X3D)
$\exists$ individual CIO c	$\exists$ widget w such as $f(c)=w$ (direct mapping)
	$\exists$ widget w such as $\forall\ c_i \in c, \exists\ f_i \mid f_i(c)=w_i$ (new mapping between properties)
	$\nexists$ widget w such as $f(c)=w$ (no possible mapping)

3. RELATED WORK

Several advanced researches could be classified according to the design space depicted in Fig. 3. Due to limited space, we only discuss those considered as the closest ones to this work.

MODIE [14] and DYNAMO-AID [4] provide a distribution manager which distributes the sub-task of a task model to various computing platforms in the same environment. This system completely supports mobilization and stationarization as the UIs could be distributed to any platform, but only one interaction surface is used at a time as a platform screen with digital objects. Therefore, the three other dimensions are not covered. This process is task based since the task model initiates the distribution of the UI fragments corresponding to the sub-tasks. Conversely, in our approach, no task model is used: each UI is first designed in an interface builder (e.g., GrafiXML – http://www.usixml.org) or imported from an external format in this editor. Each CUI resulting from this editing is then assigned to one or several interaction surface, whether it is a computer screen or not. Then, each CUI could be manipulated in direct manipulation [17] to support operations like transfer, copy, clone, duplicating, or migration.

In the MIGRATION project [1,18], partial and full migrations are supported, thus covering centralization and decentralization on top of the rest. In Everywhere [20], physicalization and digitization are added since UIs could

be rendered on various interaction surfaces such as large screens, whiteboard, wall displays as well as personal surfaces. Only digital interfaces are considered as opposed to 3DSim [19], where only hardware interfaces are supported because it is the tool's goal to prototype physical interfaces in an environment, thus also providing some sort of rapid prototyping. The material development of the physical UI is conducted after the prototype is validated. The situation is similar for [9]. XIN [12] consists of a XML-compliant language for expressing interactive systems which are distributed across various locations, with some workflow support between them. However, it does not detail much the UI structure and presentation and the world is not modeled.

In this paper, the environment model is produced as a virtual reality scene, thus allowing the rendering of both software (e.g., widgets) and hardware (e.g., physical buttons) objects, but this is achieved for rapid prototyping purposes only. It is obvious that it cannot produce a physical UI, but the corresponding UsiXML specifications could be passed to the team in charge of the development. The tool provides basic operations such as copy a CUI from one surface to another one, whether they belong to a computing platform or not, duplicate, migrate. Composition and decomposition algorithms are beyond the scope of this paper and could be found in the literature [1]. Here, they are done by graphically selecting the portions subject to (de)composition. In the next section, we exemplify the use of the design space and the tool with four representative situations where a UI is distributed.

4. FOUR SITUATIONS OF DISTRIBUTED USER INTERFACES

4.1 Across Multiple Computing Platforms

Multi-platform computing could be considered as a distribution case where a software UI is distributed successively from one platform to another one. Fig. 4 graphically depicts an example of mobilization, where a UI rendered on a monitor of a stationary PC is migrated to the screen of a highly portable platform, which can be carried by everyone. This operation is represented on the design space by laptop (top axis), one screen at a time (right axis), platform (bottom axis), and digital UI (left axis).

In order to trigger the migration, each window is augmented by an additional button freezing the UI state and allowing the user to move the window to another target, before releasing it [17]. Once migrated, the user can continue her task by manipulating the window objects. In case of migrating only a portion of it, the user can graphically select the objects subject to transfer.

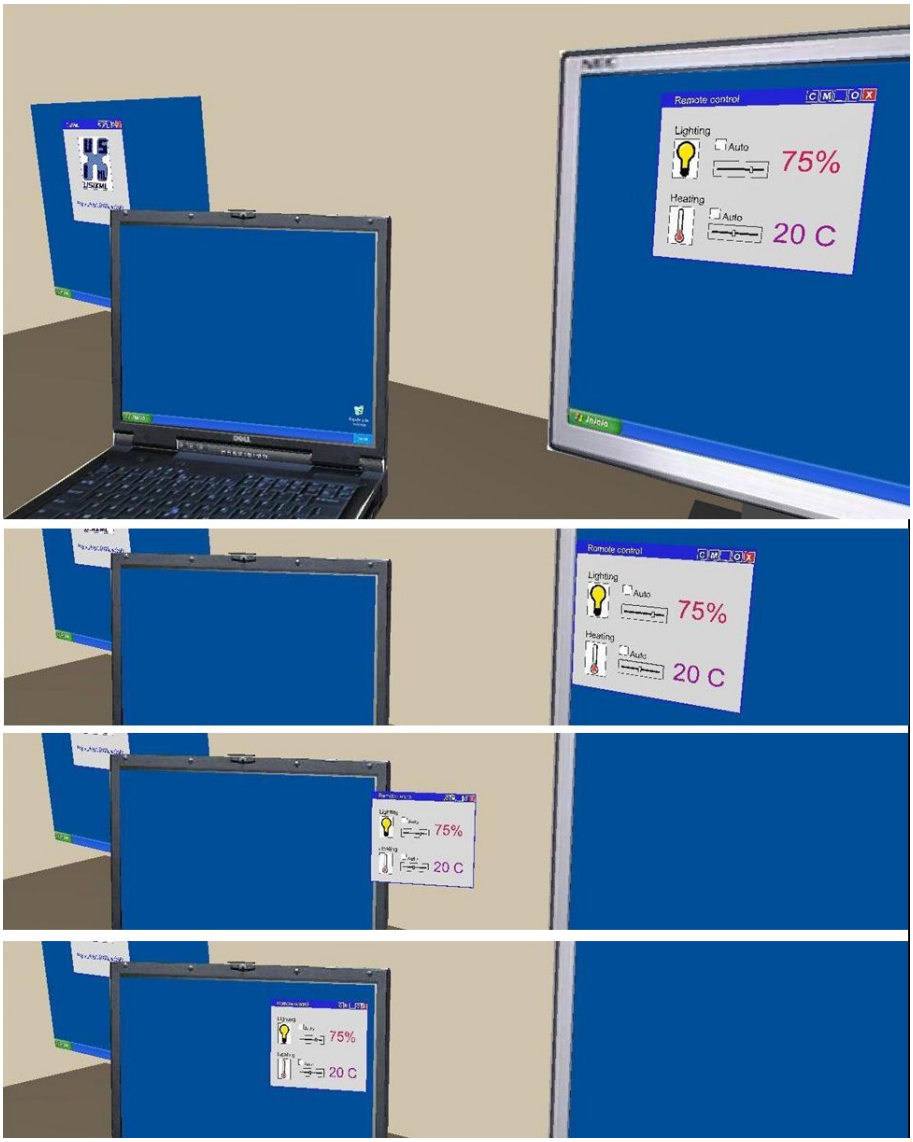

Figure 4. Example of a mobilization.

4.2 Across Multiple Interaction Surfaces

Workspace analysis could be considered as a distribution case where a software UI is distributed across multiple interaction surfaces. Workspace analysis is typically interested in investigating how information and related functions could or should be located so as to optimize the workflow, to minimize movements between locations, while allowing several persons working together. Fig. 5 graphically depicts an example where various UI

fragments are decentralized, that is, rendered on different surfaces of different computing platforms (from desktop to palmtop as in Fig. 3).

Figure 5. Example of a decentralization with different interaction surfaces.

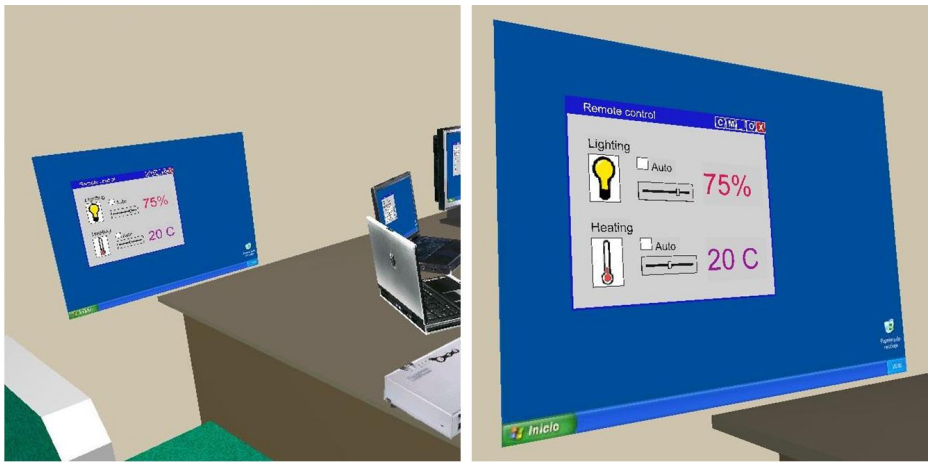

Figure 6. A software UI projected onto a wall.

4.3 Across Different Types of Interaction Surfaces

Ubiquitous or pervasive computing could be considered as a distribution case where a software/hardware UI is distributed across multiple interaction surfaces of different types. The right part of Fig. 5 demonstrates how a software UI could be distributed simultaneously on different monitors as well as an image projected onto a wall (Fig. 6 for the detailed view).

Physicalization is supported in our tool simply by moving the concerned UI from a digital interaction surface onto a physical one. The migration button is used again for this purpose, but the UI could only be released when it flies over a physical surface. Digitization is supported similarly in the inverse way. In the target interaction surface is comparable, respectively more limited in real estate than the source surface, the UI could remain unchanged, respectively be submitted to a graceful degradation process. In the case of Fig. 6, the UI remains software, thus keeping the UI definition unchanged. However, when it comes to make a UI material (through a materialization process), the UI should accommodate some changes explained in the next subsection.

4.4 Multiple Types of User Interfaces

Ambient intelligence [8,11,14,19] could be considered as a distribution case where the UI consists of software and/or hardware parts distributed in the environment. Fig. 7 exemplifies a situation where a software UI distributed across several computer screens co-exists with a physical UI embedded in the wall. In this case, the lighting and heating controls are no longer software objects, but built-in physical objects. When a software UI is submitted to materialization, the system attempts to discover a physical object which is equivalent enough in behavior to be used as a physical object. To this end, a CIO is abstracted into its corresponding Abstract Individual Component (AIC in UsiXML) at the AUI level. At this level, possible reifications of this AIC are identified and examined. If a reification corresponds directly to the initial CIO, this reified CIO is selected. Otherwise, the closest possible reification is preferred.

For instance in Fig. 7, the label displaying “75%” is transformed into a LED object displaying the same information, the heating toggle button in the software UI becomes a physical switch in the hardware UI, the “Auto” check boxes become another physical two-state switches, and the sliders become physical cursors. Ultimately, the physical objects of Fig. 7 and the software objects of Fig. 6 should be considered similar at the AUI level since they correspond to the same definition of abstract containers and individual components. Only the objects type changed. Also note that the layout changed

since the initially horizontal software controls (embedded in a horizontal box) now become vertical after performing a reshuffle of the presentation ensure by the rendering engine of the VUItoolkit.

Figure 7. A co-habitation of a software UI rendered on a screen and a physical UI rendered on a wall considered as an interaction surface.

5. CONCLUSION

In this paper, we introduced a design space for distributed user interfaces consisting of four dimensions which are supported in a tool for rapid prototyping them by direct manipulation. In addition, a pair of functions was defined for each axis to denote the progression or regression along each axis. These functions are ensured thanks to the rendering capabilities of VUIToolkit, a virtual toolkit for rendering a concrete UI specified in UsiXML. A richer environment model has been defined so as to represent the world in which the distribution may occur, thus providing a direct feedback of the configuration under study.

The various operations provided by the toolkit enable designers to explore various distributions and keeping the one which is finally found adequate to the final goals. In the future, we plan to extend this work with multiple rooms and multiple users' characterizations. "The reality of this situation (provided by this virtual representation manipulation), when you do not believe in it, refuses to disappear" (Peter Viereck).

ACKNOWLEDGEMENTS

We gratefully acknowledge the support of the SIMILAR network of excellence (http://www. similar.cc), the European research task force creating human-machine interfaces similar to human-human communication of the European Sixth Framework Programme (FP6-2002-IST1-507609). This research is fully funded by SIMILAR. The authors would like also to thank the reviewers for their comments and Gaëlle Calvary for her feedback on an early version of this manuscript.

REFERENCES

[1] Berti, S., Paternò, F., and Santoro, C., *A Taxonomy for Migratory User Interfaces*, in Proc. of 12th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2005 (Newcastle upon Tyne, 13-15 July 2005), M. Harrison (ed.), Lecture Notes in Computer Science, Vol. 3941, Springer-Verlag, Berlin, 2005.

[2] Bischofberger, W.R. and Pomberger, G., *Prototyping-Oriented Software Development–Concepts and Tools*, Springer-Verlag, Berlin, 1992.

[3] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J., *A Unifying Reference Framework for Multi-Target User Interfaces*, Interacting with Computers, Vol. 15, No. 3, June 2003, pp. 289-308.

[4] Clerckx, T., Vandervelpen, Ch., Luyten, K., and Coninx, K., *A Task Driven User Interface Architecture for Ambient Intelligent Environments*, in Proc. of 10th ACM Int. Conf. on Intelligent User Interfaces IUI'2006 (Sydney, 29 January-1 February 2006), ACM Press, New York, 2006, pp. 309-311.

[5] Coutaz, J., Lachenal, Ch., and Dupuy-Chessa, S., *Ontology for Multi-surface Interaction*, in Proc. of 9th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2003 (Zurich, 1-5 September 2003), IOS Press, Amsterdam, 2003, pp. 447-454.

[6] Demeure, A., Calvary, G., Sottet, J.-B., Ganneau, V., and Vanderdonckt, J., *A Reference Model for Distributed User Interfaces*, in Proc. of 4th Int. Workshop on Task Models and Diagrams for user interface design TAMODIA'2005 (Gdansk, 26-27 September 2005), ACM Press, New York, 2005, pp. 79-86.

[7] Dey, A.K., Salber, D., and Abowd, G.D., *A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications*, Human-Computer Interaction Journal, Vol. 16, Nos. 2-4, 2001, pp. 97-166.

[8] Florins, M., Simarro, F.M., Vanderdonckt, J., and Michotte, B., *Splitting Rules for Graceful Degradation of User Interfaces*, in Proc. of 10th ACM Int. Conf. on Intelligent User Interfaces IUI'2006 (Sydney, 29 January-1 February 2006), ACM Press, New York, 2006, pp. 264-266.

[9] Gea, M., Garrido, J.L., López-Cózar, R., Haya, P.A., Montoro, G., and Alamán, X., *Task Modelling for Ambient Intelligence*, in Proc. of 12th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2005 (Newcastle upon Tyne, 13-15 July 2005), Lecture Notes in Comp. Science, Vol. 3941, Springer-Verlag, Berlin, 2005.

[10] Grudin, J., *Partitioning Digital Worlds: Focal and Peripheral Awareness in Multiple Monitor Use*, in Proc. of ACM Conf. on Human Aspects in Computing Systems CHI'2001 (Seattle, 31 March-5 April 2001), ACM Press, New York, 2001, pp. 458-465.

[11] Gu, T., Pung, H.-K., and Qing Zhang, D., *Toward an OSGi-Based Infrastructure for*

Context-Aware Applications, Pervasive Computing, Oct.-Dec. 2004, pp. 66-74.

[12] Li, B., Tsai, W.-T., and Zhang, L.-J., *A Semantic Framework for Distributed Applications*, Proc. of the 5th Int. Conf. on Enterprise Information Systems ICEIS'2003 (Angers, 22-26 April 2003), Volume IV - Software Agents and Internet Computing, pp. 34-41.

[13] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Lopez, V., *UsiXML: a Language Supporting Multi-Path Development of User Interfaces*, in Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 200-220.

[14] Luyten, K., Vandervelpen, Ch., and Coninx, K., *Task Modeling for Ambient Intelligent Environments: Design Support for Situated Task Executions*, Proc. of 4th Int. Workshop on Task Models and Diagrams for user interface design TAMODIA'2005 (Gdansk, 26-27 September 2005), ACM Press, New York, 2005, pp. 87-94.

[15] Lyytinen,K. and Yoo, Y., *Issues and Challenges in Ubiquitous Computing*, Communications of the ACM, Vol. 45, No. 12, 2002, pp. 62-65.

[16] Molina, J.P., Vanderdonckt, J., Montero, F., and Gonzalez, P., *Towards Virtualization of User Interfaces based on UsiXML*, in Proc. of 10th ACM Int. Conf. on 3D Web Technology Web3D'2005 (Bangor, 29 March-1 April 2005), ACM Press, New York, 2005, pp. 169-178.

[17] Molina, J.P., Vanderdonckt, J., and González, P., *Direct manipulation of User Interfaces for Migration*, in Proc. of 10th ACM Int. Conf. on Intelligent User Interfaces IUI'2006 (Sydney, 29 January-1 February 2006), ACM Press, New York, 2006, pp. 140-147.

[18] Mori, G. and Paternò, F., *Automatic Semantic Platform-dependent Redesign*, in Proc. of Joint sOc-EUSAI'2005 (Grenoble, October 2005), pp. 177-182.

[19] Nazari Shirehjini, A.A., Klar, F., and Kirste, T., *3DSim: Rapid Prototyping Ambient Intelligence*, in Proc. of the 2005 Joint Conf. on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies sOc-EUSAI'2005 (Grenoble, October 2005), ACM Int. Conf. Proc. Series, Vol. 121, 2005, pp. 303-307.

[20] Pinhanez, C., *The Everywhere Displays Projector: A Device to Create Ubiquitous Graphical Interfaces*, in Proc. of the 3rd Int. Conf. on Ubiquitous Computing UbiComp'2001 (Atlanta, 30 September- 2 October 2001), Lecture Notes in Computer Science, Vol. 2201, Springer-Verlag, Berlin, pp. 315-331.

[21] Tan, D.S. and Czerwinski, M., *Effects of Visual Separation and Physical Discontinuities when Distributing Information across Multiple Displays*, in M. Rauterberg, M. Menozzi, J. Wesson (eds.), Proc. of 9th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2003 (Zurich, 1-5 September 2003), IOS Press, Amsterdam, 2003, pp. 9-16.

[22] Vanderdonckt, J., Bouillon, L., Chieu, C.K., and Trevisan, D., *Model-based Design, Generation, and Evaluation of Virtual User Interfaces*, in Proc. of 9th ACM Int. Conf. on 3D Web Technology Web3D'2004 (Monterey, April 5-8, 2004), ACM Press, New York, 2004, pp. 51-60.

[23] Vanderdonckt, J. and Bodart, F., *Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection*, in Proc. of the ACM Conf. on Human Factors in Computing Systems INTERCHI'93 (Amsterdam, 24-29 April 1993), ACM Press, New York, 1993, pp. 424-429.