Distributed User Interfaces: How to Distribute User Interface Elements across Users, Platforms, and Environments

Jean Vanderdonckt
Université catholique de Louvain, Louvain School of Management, Louvain Interaction Laboratory
Place des Doyens, 1 – B-1348 Louvain-la-Neuve, Belgium
jean.vanderdonckt@uclouvain.be

Abstract

Distributed User Interfaces (DUIs) have become one vivid area of research and development in Human-Computer Interaction (HCI) where many dramatic changes occur in the way we can interact with interactive systems. DUIs attempt to surpass user interfaces that are manipulated only by a single end user, on the same computing platform, and in the same environment, with little or no variations among these axes. In contrast to such currently existing user interfaces, DUIs enable end users to distribute any user interface element, ranging from the largest one to the smallest one, across one or many of these dimensions at design- and/or run-time: across different users, across different computing platforms, and across different physical environments. In this way, end users could be engaged in distributed tasks that are regulated by distribution rules, many of them being currently used in the real world. This paper provides a conceptual framework that invites us to rethink traditional user interfaces in a distributed way based on the locus of distribution control: in the hands of the end user, under control of the system, or in mixed-initiative way. Any user interface submitted to distribution may also be subject to adaptation with respect to the user, the platform, and the environment.

1. Introduction

If we look back retrospectively to the evolution of concerns in Human-Computer Interaction (HCI) from a Software Engineering (SE) point of view, we can observe that several models appeared over time in order to address the shortcomings observed in the previous generation of models. For instance, a data model has been progressively replaced by a domain model in order to automate User Interface (UI) generation because the data model was considered not enough expressive: data structure were almost flat, data type are under-specified, semantic relationships were absent, constraints are not explicitly formalized, etc. Today, we reached a point where the prevalent models used to characterize a UI are task, domain, abstract UI, concrete UI, and final UI, if we consider for example the Cameleon Reference Framework (CRF) [6]. Behind the curtains, this framework assumes that only *one context of use is considered at a time*. By context of use, we hereby understand that one user is carrying out her task on a dedicated computing platform in a given environment, thus leading to one single context. A context is again considered as a triple $C=(U,P,E)$ where U denotes a user model, P denotes a platform model, and E denotes an environment model.

The consideration of one context of use at a time is today completely surpassed by existing situations in the real world: a given user is rarely working alone and is largely involved in cooperation and collaboration; a user is rarely using one single platform at a time, but several different platforms at a time or one after another, and a user is no longer staying in the same environment since she is moving from one environment to another or across environments. In addition, a same task is no longer carried out by a single user, but by a multitude of different users, simultaneously or not. All these reasons stem for considered the fact that a UI is *no longer concentrated, but distributed across users, platforms, and environments*, the three main dimensions of UI distribution.

In this paper we tackle this problem with the introduction of a transversal model for expressing a Distributed User Interface (DUI) that supports the aforementioned considerations. Section 2 introduces our operational DUI definition and then discusses some principles for distributing a UI along these three dimensions and exemplifies them on examples taken from the state of the art, one facet at a time: task, domain, user interface, user, platform, and environment respectively. Section 3 introduces some principles for DUIs. Section 4 shows some conclusions and future work.

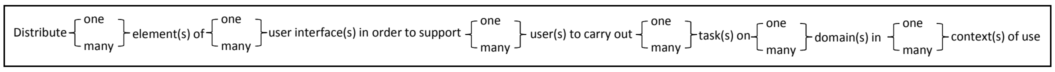


Figure 1. A transversal model of DUI.

2. Terminology for DUIs

The literature abounds in the usage of different terms for expressing different situations of distribution [5,12,14,18,20], some of them being synonyms, some of them radically different, thus posing a problem of a consensual ontology in the domain. The following distributions could occur:

- *Multi-monitor usage*: a single user using a single computing platform wants to distribute her UI across various monitors connected to the same platform [10]. For instance, a dual display is supported when a graphic card expands the main monitor of a computing platform to two or more connected monitors.
- *Multi-device usage*: a single user uses several different devices together, whether they are running the same operating system or not [22]. For instance, a user controls a music player running on a media center using a remote control running on a handheld Personal Digital Assistant (PDA) and/or on a physical device.
- *Multi-platform usage*: a single user uses heterogeneous computing platforms, perhaps running different operating systems [8]. Multi-device usage subsumes a multi-platform usage (since there are different machines) but the reciprocal does not hold: a user could use several computers (hence, multi-platform) that are similar (hence, no multi-device) [25,26].
- *Multi-display usage*: we hereby define multi-display as a combination of multi-monitor and multi-device usages [22]. A single user may distribute a UI across multiple monitors and devices simultaneously [24].
- *Multi-user*: it represents an extension of the previous usages to multiple users concurrently [5]. In this case, one or many users may want to distribute parts or whole of their UI across several monitors, devices, platforms, or displays. For instance, in a control room setup, users may want to direct portions of a UI to other displays of others users depending on the context of use. When a multi-user interface is of concern, it is also typically used for supporting tasks that are allocated or de-allocated from one user to another one, such as in task delegation, task suspension and resuming.

All these terms refer to some particular case of a DUI. Depending on the source, the terms found in the literature sometimes refers to the same situation, sometimes not. Therefore, we believe that in order to introduce a correct definition of a situation for a DUI, there is a need to define a model of distribution that is transversal to the different levels of abstraction that are typically found in a User Interface Development Life Cycle (UIDLC).

Let us consider the field of distributed computing [27]: "a distributed system consists of multiple autonomous computers that communicate through a computer network". If we extrapolate a DUI definition from this definition, this would give "a distributed user interface consists of multiple autonomous user interfaces that communicate through a computer network". This definition is very much reduced in that it does not consider several aspects to be considered in a distribution: task, domain, abstract or concrete UI, context of use, which is in turn decomposed into platform, user, and environment. To overcome these shortcomings, we suggest a transversal model (Fig. 1):

A *UI distribution* concerns the repartition of one or many **elements** from one or many **user interfaces** 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.

These different aspects are elaborated further in the following respective sub-sections. The inverse operation is defined as follows:

A *UI federation* concerns the concentration of one or many elements from one of many user interfaces 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.

Two significant cases of UI federation exist: after a UI distribution has been operated, a UI federation may be triggered in order to restore the initial situation; if a UI federation is triggered not after a UI distribution, then a UI composition may be triggered depending on the conditions imposed in the federation.

2.1. Distribute what? The elements

At first glance, the atomic element that could be submitted to distribution is any UI widget, whether it is native or not in a toolkit, API, or programming environment. However, if we even consider that a widget is itself composed of other elements (for instance, a radio box is composed of mutually exclusive circles and radio items), then any of these individual elements could be also submitted to distribution (for instance, a radio circle could be separated from its radio item). Most UI toolkits do not natively support this distribution, thus requiring a manual overwriting of the expose methods for these widgets. Lower than any final element is the pixel: in principle, this is the most atomic level where distribution may occur.

For this purpose, several techniques exist: physical, logical, and semantic pixel conservations [7]. If a UI that consumes S.L x S.H pixels should be distributed on a surface D witch is D.L x D.H pixels, then we need to consider that pixel size on S is S.Pix and pixel size on D is D.Pix. Fig. 2 illustrates possibilities in the case D.Pix / S.Pix > 1. The more the case is located in the top right corner, the better it is.

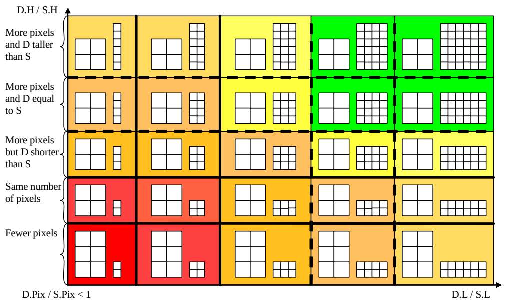

D.H / S.H
More pixels and D taller than S
More pixels and D equal to S
More pixels but D shorter than S
Same number of pixels
Fewer pixels
D.Pix / S.Pix < 1
D.L / S.L

Figure 2. Different situations at the pixel level.

Another representative example is the WinCuts system [23] that augments window managers by letting users acquire and interact with alternative views of arbitrary regions of existing windows. The Frisbee [13] is a widget that acts as a telescope to a remote area on the display (http://www.autodeskresearch.com/publications/frisbee). Users manipulate remote items by interacting with their proxies within the Frisbee's main area and reposition items on the main display by moving them through specified transfer channels.

2.2. Distribute from what? The user interfaces

All elements subject to UI distribution should of course belong to one or many UIs that should be clearly identified. So far, we have considered that all UI are graphical or at least have some graphical feedback, even if other interaction modalities are involved. For instance, if a multimodal UI involves vocal and tactile interaction modalities, a graphical modality could be added in order to provide the end user with some feedback about the task being carried out. Theoretically speaking, other interaction modalities could be also distributed, but this is another research to be conducted. For instance, speech syntheses and recognitions could be distributed across several platforms not only to optimize the computational power, but also in order to help differentiating the speakers.

2.3. Distribute for which? The tasks

So far, the task has often been considered unique in a single context of use or multiple contexts of use, thus raising some variations in order to address the constraints imposed by the different contexts of use. In order to be fully distributed, one or many tasks should be considered to be carried out simultaneously or not in a distributed way. In the field of ambient intelligence, Luyten *et al*. [17,18] introduced the notion of situated task in order to model how a task could be distributed into several sub-tasks to be carried out by one user, but on different platforms in the same environment over time (Fig. 3). This is a very important way to represent one task that should be generalized to any amount of tasks, whether they are carried out by one or many users.

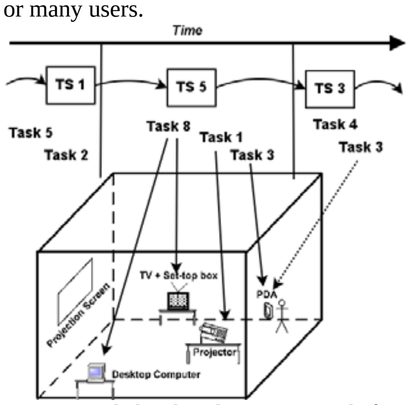


Figure 3. A task distributed across many platforms. (Reproduced from [17] with permission).

In the field of workflow information systems, FlowiXML introduced a series of workflow UI patterns that address several management patterns for distributing tasks, ranging from simple delegation to offering to one or many candidates [11]. For instance, a task could be offered to one or many resources, one of which accepting it, carrying out it and returning the results to the initiator.

2.4. Distribute on what? The domains

This aspect is very much relevant to the computer science field of distributed databases. "A distributed database is a database that is under the control of a central DataBase Management System (DBMS) in which storage devices are not all attached to a common CPU. It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers. Collections of data (e.g., in a database) can be distributed across multiple physical locations. A distributed database can reside on network servers on the Internet, on corporate intranets or extranets, or on other company networks. Replication and distribution of databases improve database performance at end-user worksites." (Source: http://en.wikipedia.org/wiki/Distributed_database). So far, we were used to model one single task attached to one domain model, but this could be generalized to one task model attached to one or several (potentially distributed) domain models.

2.5. Distributed abstract/concrete UIs

In Sub-section 2.2, we simply mentioned the distribution of a Final UI, as defined in the CRF [6]. This principle could therefore be propagated to any other upper level, such as Concrete User Interface (CUI) level and Abstract User Interface (AUI) level.

2.6. Distribute across what? The platforms

This dimension has probably received the largest attention since the platform is certainly one parameter that significantly influences the design of DUIs. Significant progress has been in the area of multi-device UIs (where UIs are produced for several devices simultaneously) and in UI migration (where UIs are migrated from one device to another while maintaining task continuity). Less work has been however devoted towards dividing a UI across devices, displays, or platforms, where they are used by the same user or shared by different users [2,3]. During the last decade, a DUI was mostly defined in terms of platform distribution: a DUI was defined as any application UI whose elements can be distributed across different displays, devices, or different computing platforms. 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 with the interaction capabilities that are constrained on this display/device/platform [5].

DUIs have been subject to several studies that investigate their specific abilities with respect to platform distribution that may lead to design implications. This includes use of multiple monitors on a same computing platform by a single user [10], use of multiple platforms by a single user with synchronisation between, exchange of information between platforms belonging to different users, moving information between displays on a single platforms, partition of tasks across displays for a single user [1], sharing common information on a common display while keeping some information private on a own platform,…

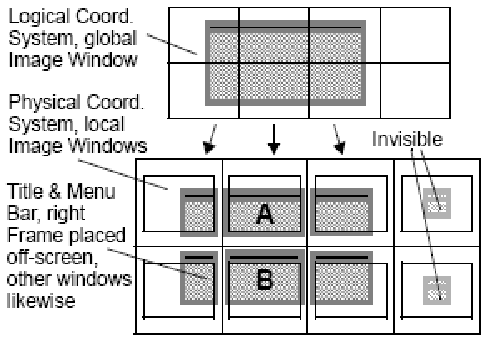

Logical Coord. System, global Image Window
Physical Coord. System, local Image Windows
Title & Menu Bar, right Frame placed off-screen, other windows likewise
Invisible
A
B

Figure 4. Partitioning of a window across several screens to obtain a DUI [7].

Partitioning a window across several screens, at the physical or at the logical level, is not comparable and involve different systems of coordinates (Fig. 4). Several early techniques have been implemented in order to support multi-display distribution such as, but not limited to: X11 remote displays (www.x.org/), Virtual Network Computing (VNC - www.uk.research.att.com/vnc/), and

Windows Terminal Server (http://www.microsoft.com/windows2000/technologies/terminal/default.asp), all allowing a window to move from one screen to another at the window manager level. But this is not the same level as application migration across workstations [2] or task-oriented migration of parts or whole of the UI [2].

Beale & Edmondson [4] conducted user surveys 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.

Tan & Czewinsky [23] found out 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 (e.g., a tablet) to another one (e.g., a tabletop) in four different distance ranges and with three movement directions. Their study suggests that spatial manipulation of data was faster than pressure-based techniques.

One 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 describing them in a formal way [2]. This will enable the underlying system to decide where different DUI elements should be placed in locations that are significant and usable for a distributed task to take place.

AttachMe/DetachMe [9] is a typical DUI example where one single user distributed UI elements across several platforms at run-time, possibly running different operating systems, in order to better accomplish a given interactive task. One significant application of this interaction technique is the painter's palette [9]: in order to maximise the screen real estate used for painting, all toolbars (e.g., with paint brushes, color palettes) are dynamically migrated from the main desktop to an external platform, typically a PocketPC. In the implementation described in [9], three operating systems are supported: Mac OS X, Windows Vista, and Linux. The same distribution occurs independently of any platform and operating system, provided that the master platform is connected to the slave platform through a network connection (e.g., LAN, Wi-Fi). Let us exemplify how the AttachMe/DetachMe technique can be used for distributing an initial UI (Fig. 5) into several elements for entering information about a movie.

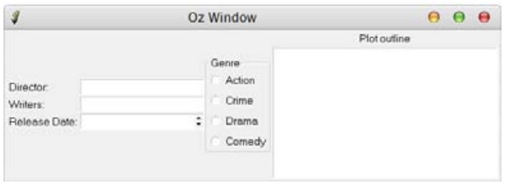


Figure 5. Initial UI to be distributed.

Let us assume that two situations should be supported: one for use on a PC where a large screen and a keyboard are present and one for use on a PDA where the screen estate is small and there is no keyboard. In the <u>first situation</u>, the PC version offers a side by side presentation for the three groups of input fields (Fig. 6).

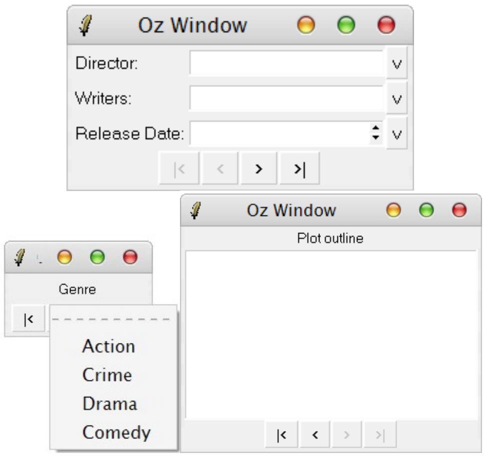


Figure 6. First DUI in three parts.

The left part uses the normal text and number input widgets. The middle part uses a set of radio buttons for selecting the genre. The right part uses a normal text widget.

In the second situation, the PDA version only displays one part at a time, with navigational buttons at the bottom (Fig. 7). Furthermore, the text and number input widgets have an arrow that displays a virtual keyboard for entering the data. The middle part uses a menu to select between the items instead of radio buttons. This widget also has a list box renderer that we could use instead. And finally the right part uses a text widget for which no alternate renderer is currently provided. If a renderer was created with support for a virtual keyboard, then we could use it by specifying it in a mechanism called "AdaptationMap", without changing any other part of the code.

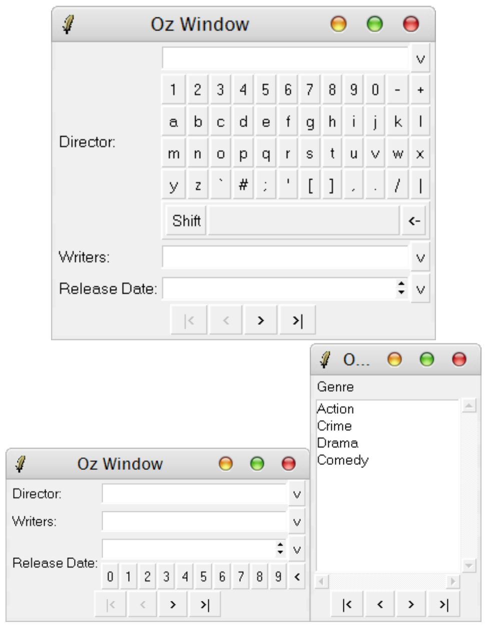


Figure 7. Second DUI in three parts.

2.7. Distribute for who? The users

It is equally important for DUI to recognize that they are used by different users, whether they are working at the same place (co-located) or not (remote collaboration [1], cooperation, competition or coopetition). For instance, the game of Pictionary is a typical example of a task distributed for many users: one player selects a word from a dictionary, a second player draws this word on an interactive 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 the points.

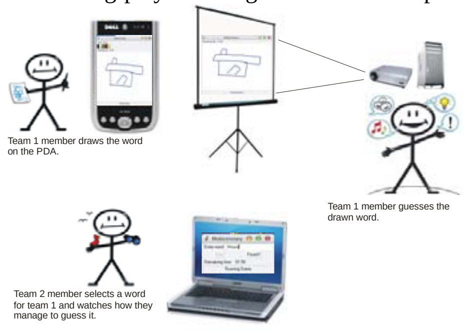
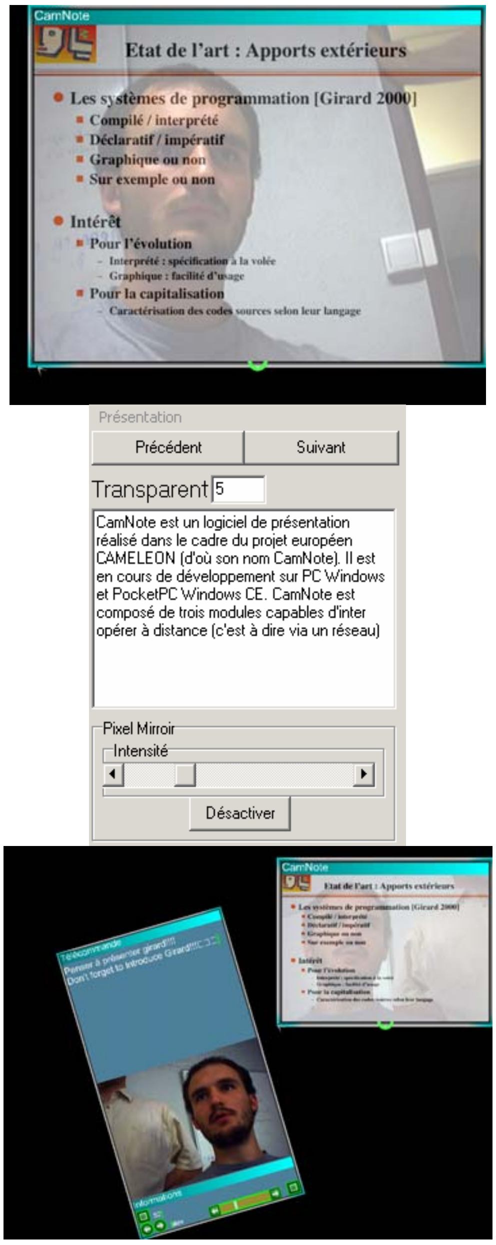

Team 1 member draws the word on the PDA.
Team 1 member guesses the drawn word.
Team 2 member selects a word for team 1 and watches how they manage to guess it.

Figure 8. Setup of the Pictionary [19].

2.8. Distribute where? The environments

An environment is often considered as the social and physical setup in which a user is carrying out a task. Changing the environment in which a task is performed may significantly affect the task performance. For instance, an office environment may provide quiet, stable, and reliable conditions to properly perform a task while a mobile environment in a corridor of this office may induce noisy, moving, and unreliable conditions. So, even if the user, the task, and the platform remain constant, the user performance may be significantly affected by a changing environment that should be in principle reflected in some change in the UI. Such a change is typically a UI distribution: when the environment changes, the end user may want to change the UI configuration by keeping only vital elements that are critical for conducting the task to its full completion, even if the details are not known or manipulated. This question is also related to the domain of *situation engineering*, in which the end user's behavior is studied through various psychological and ethnographic methods to understand how this behavior is influenced by the social, psychological, perhaps organizational aspects of the environment. In particular, situation engineering is aimed at identifying the right paths for conducting a task successfully and the bad paths that may induce task failure in order to avoid them.


CamNote

Etat de l'art : Apports extérieurs

- Les systèmes de programmation [Girard 2000]
  - Compilé / interprété
  - Déclaratif / impératif
  - Graphique ou non
  - Sur exemple ou non
- Intérêt
  - Pour l'évolution
    - Interprété : spécification à la volée
    - Graphique : facilité d'usage
  - Pour la capitalisation
    - Caractérisation des codes sources selon leur langage

Présentation

Précédent | Suivant

Transparent 5

CamNote est un logiciel de présentation réalisé dans le cadre du projet européen CAMELEON (d'où son nom CamNote). Il est en cours de développement sur PC Windows et PocketPC Windows CE. CamNote est composé de trois modules capables d'inter opérer à distance (c'est à dire via un réseau)

Pixel Mirroir

Intensité

Désactiver

Télécommande

Penser à présenter girard!!!

Don't forget to introduce Girard!!!

Informations

Figure 9. Setup of the Pictionary [?].

Figure 9 exemplifies a system where a UI distribution is triggered by a change of environment. In a typical presentation environment, a presenter would typically present slides by browsing through the slides via a presentation software on a centralized platform. A physical remote control device could support her for some limited actions, such as “next slide”, “previous slide”. When a presentation is conducted remotely, the presenter is willing to embed a video streaming of her. In order to avoid having two separate screens that consume screen real estate, transparency could be used to superimpose the presenter’s video on top of the slides. So, when the presenter is in the room (one environment), a normal control configuration is desired; when the presenter is remote (in another environment), a superimposed control configuration is preferred. Figure 9 depicts the whole system when UI distribution occurs between a PC and a PDA. Note that only functional core is distributed. Indeed, native UI descriptions on a PC and on a PDA are quite different. Using a same toolkit for both platforms running in the different environments would have been more complex to implement. When the PDA disappears, the remote controller function is migrated to the PC. When then system is re-centralized, the window containing the remote controller could be merged with the slide show or kept separated. When this operation occurs, the window is being rotated in Figure 9 in order to animate the transitions between the two situations: *centralized* (one environment) and *decentralized* (two environments).

In [28], Vandervelpen *et al.* present a lightweight system for distributing services to different users who are in a same physical environment, but who may assume different roles. This interesting setup raises the question of expanding the definition of environment to a situation, where a situation is a particular configuration of an environment in which some users are assigned to particular roles. In the setup described in [28], different people are being assigned different services, e.g., “next slide”, “previous slide”, “zoom in”, “zoom out” depending on the role they want to play in the presentation. Each service is assigned to one single user on one platform at a time, but one can easily imagine that the same service could be reproduced and distributed several times to different users or to the same users if she is using different platforms.

Speakeasy consists of a computing framework that is designed to support use of resources such as displays/devices/platforms that appear/disappear opportunistically, called recombinant computing [20] depending on the environment.

3. Design Principles for DUIs

After having examined the various dimensions along which UI distribution may occur, this section suggests some instantiations of the transversal model of DUI introduced in Fig. 1 by means of design principles. These design principles are introduced in order to address current design shortcomings of the typical situation discussed in the introduction: one single user carrying out one single task in one unique context of use. It is true that these shortcomings are intertwined, as we observed in Section 2 that each instantiation of the Fig. 1 immediately creates interdependencies between the various dimensions covered.

Each UI distribution could be also interpreted as a form of UI adaptation since an original UI is transformed into a target UI in order to be adapted to a new situation. As such, UI distribution could be considered as a particular form of UI adaptation, but not vice versa.

3.1. Design Principle for distributing tasks

The instantiation of the transversal model of Fig. 1 in this case gives: *distribute one UI in order to support one user in carrying out one task on one domain in many contexts of use*. The end user should be empowered with UI distribution mechanisms in order to carry out the same task while the context of use is changing: from one platform to another or from one environment to another. If the task remains constant, the UI distribution should also help the end user in requesting help to other users for ensuring the successful completion of this task or in allocating any sub-task to another user. A sub-task of a main task could be delegated to another user because of lack of familiarity, expertise, resource, time, availability of the primary end user. Or for reasons that are external to the primary end user: responsibilities, jobs definition, separation of duties, role-based allocation, round robin, etc. Actually, any pattern for task delegation is applicable. In this way, multiple work methods for carrying out the same task become possible. For instance, a particular section of a complex form could be sent to another user who is more expert for this section than the initiator, before returning it filled in to the initiator (i.e. the person who requested the help to the secondary user).

3.2. Design Principle for distributing users

The instantiation of the transversal model of Fig. 1 in this case gives: *distribute one UI in order to support many users in carrying out many tasks on one domain in one context of use*. In other words, when a particular context of use is given, several users should be able to apply UI distribution to their respective UIs (perhaps one UI if the system is centralized or many if the system is decentralized or if multiple systems are available) in order to carry out one common task that could be decomposed into sub-tasks that are under the responsibility of different users. Again, organisational allocation of tasks and related sub-tasks should be applied in order to determine how these sub-tasks will be offered to one or multiple users, will be allocated to one or multiple users, and will be executed by one or multiple users. In organisational allocation of tasks, the system should exhibit the ability to offer or allocate a task to users based on their position within the organization and their relationship with other users. Alternative work methods could also be investigated such as: allocation of a task to one or many users based on any attribute or property of the user, the task, and the relationship between. This includes allocation of a task to a user based on experience, history, success rate, familiarity with the type of task.

3.3. Design Principle for distributing platforms

The instantiation of the transversal model of Fig. 1 in this case gives: *distribute many UIs in order to support one user in carrying out many tasks on one domain in one environment*. In other words, given that the user, the task, and the environment remain constant, the platforms capabilities should be investigated in order to enable the end user to optimize the carrying out of one or many tasks based on the specific properties of these platforms, and their suitability for a task or any of its sub-tasks. The example of the painter’s palette is obvious: two platforms are available to the end user for drawing and the global screen real estate should be exploited in order to maximize the task performance. Fig. 10 illustrates this design principle: three toolbars are progressively extracted from the main UI of a vector drawing application, recomposed to form a set of related palettes. Here the UI distribution is reproduced on the same platform for the simplicity of the screenshots, but the new set of palettes could reside on any other platform of the user, provided that it is connected through a computer network.

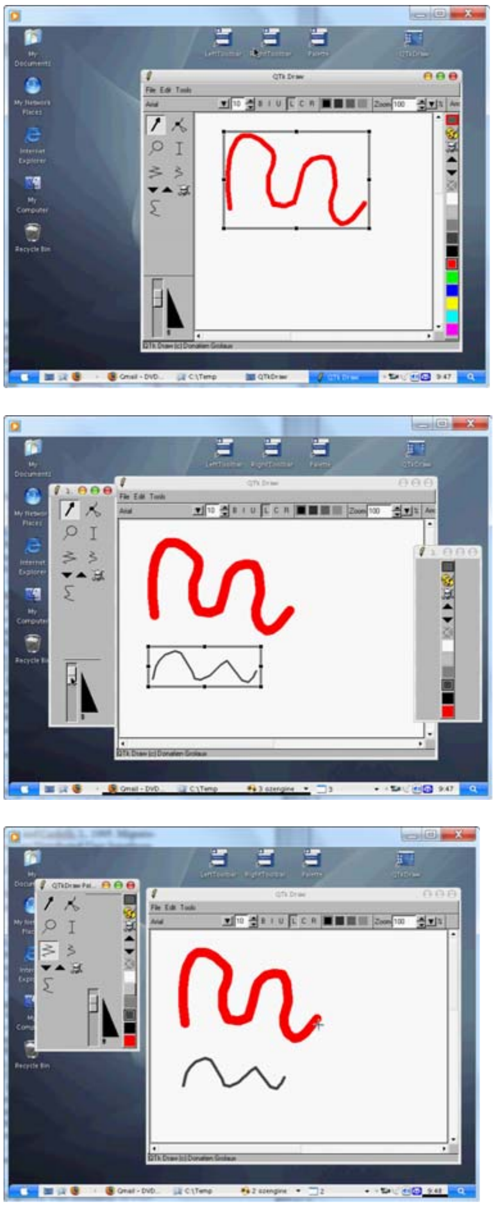


Figure 10. Three different UI distributions for the painter's palette, here on the same platform [19].

The problem of screen allocation to (sub-)tasks could be solved by a multi-criteria approach if all the properties of concern are known at runtime with their corresponding value. But this would give a system with automated UI layout [16] across several platforms depending on the constraints imposed by these platforms.

3.4. Design Principle for distributing environments

The instantiation of the transversal model of Fig. 1 in this case gives: *distribute one UI in order to support one user in carrying out many tasks on one domain in many environments*. In other words, the end user should be empowered with UI distribution mechanisms that enable her to conduct the same task but in different ways in different environments, while taking into account the properties of each environment. This does not necessarily include the usage of several platforms, but in case of their availability, this should be taken into account. Based on situation engineering, this design principle is intended to support the multiple behaviors that could be produced by a single user in different environments or situations. The UI distribution should produce a UI configuration that is adjusted to the particular constraints or properties of one or many environments, including the smooth transition from one environment to another.

4. Conclusion

In this paper, we introduced a transversal model for expressing a distributed user interface based on the different aspects that may influence the success of a distribution: elements, type of UI, task, domain, abstract or concrete UI, user, platform and environment. Each aspect becomes a dimension along which design principles could be elaborated in order to overcome the current limitations imposed by the stereotyped UI of: *one user is carrying out one task on one domain in a fixed context of use*. This stereotyped UI is no longer applicable today due to permeable boundaries of users (i.e. a user may participate in different roles in different groups), platforms (i.e. a platform is itself included in a cluster, in a larger infrastructure), and environments (i.e. an environment could give rise to different situations and different environments could be easily connected to each other).

Only the problem of multiple domains has not been explicitly addressed in this paper since it is more relevant to the field of distributed data bases. In this field, several domain models co-exist that are interrelated based on high-level relationships that could give rise to another model. For instance, several distributed entity-relationship-attribute (ERA) models that could serve to capture a domain could co-exist and be interrelated into a new higher-level ERA model. This is considered beyond the scope of this paper.

In the near future, we plan to address the aforementioned design principles for DUIs by developing a UI toolkit that provides the developer with distribution primitives that are context independent. These primitives could then be called in a distribution scenario that explicitly represents the logic of a UI distribution based on the transversal model introduced in Fig. 1.

By this model and these design principles, we encourage any research and development to investigate to what extent they could be supported by appropriate interaction techniques and user studies to determine to what extent the availability of these techniques for supporting UI distribution induce a significant effect on the end user, perhaps on end user satisfaction, task performance, or any other relevant metric. These aspects will be integrated in the UsiXML [15] V2.0 User Interface Description Language.

Acknowledgments

The author would like to thank Óscar Pastor (Universidad Politécnica de Valencia), Pedro Latorre (Universidad de Zaragoza), and José Luis Garrido (Universidad de Granada) for inviting us to give this keynote address. We also acknowledge the support of the FP7 Serenoa project funded by the European Commission.

References

[1] Arroyo, R.F., Gea, M., Garrido, J.L., Haya, P.A. Development of Ambient Intelligence Systems Based on Collaborative Task Models. Journal of Universal Computer Science 14, 9 (2008), pp. 1545-1559.

[2] Bandelloni, R. and Paternò, F. Migratory user interfaces able to adapt to various interaction platforms. International Journal of Human-Computer Studies 60, 5-6 (2004), pp. 621-639.

[3] Bharat, K.A. and Cardelli, L. Migratory Applications Distributed User Interfaces. Proc. of ACM Symposium on User Interface Software Technology UIST'95 (Pittsburgh, Nov. 1995). ACM Press, New York, 1995, pp. 132-142.

[4] Beale, R. and Edmonson, W. Multiple Carets, Multiple Screens and Multi-Tasking: New Behaviours with Multiple Computers. Proc. of 21st British CHI Group Annual Conference on Human-Computer Interaction HCI'2007 (Lancaster, September 3-7, 2007). British Computer Society, 2007, pp. 55-64.

[5] Berglund, E. and Bång, M. Requirements for distributed user interface in ubiquitous computing networks. Proc. of 1st Int. Conf. on Mobile and ubiquitous multimedia MUM'2002. ACM Press, New York, 2002

[6] Calvary, G., J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt, A Unifying Reference Framework for Multi-Target User Interfaces, Interacting with Computers 15, 3 (June 2003), pp. 289-308.

[7] Demeure, A., Sottet, J.S., Calvary, G., Coutaz, J., Ganneau, V., Vanderdonckt, J.. The 4C Reference Model for Distributed User Interfaces. Proc. of 4th Int. Conf. on Autonomic and Autonomous Systems ICAS'2008 (Gosier, 16-21 March 2008), D. Greenwood, M. Grottke, H. Lutfiyya, M. Popescu (eds.). IEEE Computer Society Press, Los Alamitos, 2008, pp. 61-69.

[8] Grolaux, D., Van Roy, P., and Vanderdonckt, J. Migratable User Interfaces: Beyond Migratory User Interfaces. Proc. of 1st IEEE-ACM Annual Int. Conf. on Mobile and Ubiquitous Systems: Networking and Services MOBIQUITOUS'04 (Boston, August 22-25, 2004). IEEE Computer Society Press, Los Alamitos, 2004, pp. 422-430.

[9] Grolaux, D., Vanderdonckt, J., and Van Roy, P. Attach me, Detach me, Assemble me like You Work. Proc. of IFIP Conf. on Human-Computer Interaction INTERACT'05 (Rome, 12-16 September 2005). Lecture Notes in Computer Science, Vol. 3585, Springer-Verlag, Berlin, 2005, pp. 198-212

[10] Grudin, J. Partitioning digital worlds: focal and peripheral awareness in multiple moni-

tor use. Proc. of ACM Conf. on Human Aspects in Computing Systems CHI’01 (Seattle, March 2001). ACM Press, New York, 2001, pp. 458-46.

[11] Guerrero, J., Vanderdonckt, J., Gonzalez, J. FlowiXML: a Step towards Designing Workflow Management Systems, Journal of Web Engineering 4, 2 (2008), pp. 163-182.

[12] Hutchings, H.M. and Pierce, J.S. Understanding the Whethers, Hows, and Whys of Divisible Interfaces. Proc. of ACM Working Conf. on Advanced Visual Interfaces AVI’06 (Venezia, May 23-26, 2006). ACM Press, New York, 2006, pp. 274-277.

[13] Khan, A., Fitzmaurice, G., Almeida, D., Burtnyk, N., and Kurtenbach, G. A Remote Control Interface for Large Displays. Proc. of ACM Conf. on User Interface Software Technology UIST 2004. ACM Press, new York, 2004, pp. 127-136.

[14] Larsson, A. and Berglund, E. Programming ubiquitous software applications: requirements for distributed user interfaces. Proc. of 16th International Conference on Software Engineering & Knowledge Engineering SEKE'2004 (Banff, June 20-24, 2004). ACM Press, New York, 2004, pp. 246-251.

[15] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Lopez, V. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. Proc. of 9th IFIP Conf. on Engineering for HCI (Hamburg, 2004). Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 207-228.

[16] Limbourg, Q., Vanderdonckt, J. Multi-Path Transformational Development of User Interfaces with Graph Transformations, in Seffah, A., Vanderdonckt, J., Desmarais, M. (eds.), “Human-Centered Software Engineering”, Chapter 6, HCI Series, Springer, London, 2009, pp. 109-140.

[17] Luyten, K. and Coninx, K. 2005. Distributed User Interface Elements to support Smart Interaction Spaces. Proc. of the 7th IEEE Int. Symposium on Multimedia, IEEE Comp. Society, Washington, DC, pp. 277-286.

[18] 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.

[19] Melchior, J., Grolaux, D., Vanderdonckt, J., Van Roy, P., A Toolkit for Peer-to-Peer Distributed User Interfaces: Concepts, Implementation, and Applications. Proc. of 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems EICS’2009 (Pittsburgh, July 15-17, 2009). ACM Press, New York, 2009, pp. 69-78.

[20] 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. Proc. of ACM Conf. on User Interface Software Technology UIST'2002. ACM Press, New York, 2002, pp. 171-180.

[21] Pastor, O., Gómez, J., Insfrán, E., Pelechano, V., The OO-Method approach for information systems modeling: from object-oriented conceptual modeling to automated programming, Information Systems 26, 7 (November 2001), pp. 507-534.

[22] Sjölund, M., Larsson, A., and Berglund, E. Smartphone Views: Building Multi-device Distributed User Interfaces. Proc. of Conf. on Mobile Human-Computer Interaction MobileHCI'2004. Lecture Notes in Computer Science, Volume 3160, Springer-Verlag, 2004, pp. 127-140.

[23] Tan, D.S. and Czerwinski, M. Effects of Visual Separation and Physical Discontinuities when Distributing Information across Multiple Displays. Proc. of 9th IFIP TC 13 Int. Conf. on Human-Computer Interaction Interact’2003 (Zurich, 1-5 September 2003), M. Rauterberg, M. Menozzi, J. Wesson (eds.), IOS Press, Amsterdam, 2003, pp. 252-260.

[24] Vanderdonckt, J., Furtado, E., Furtado, V., Limbourg, Q., Silva, W., Rodrigues, D., Taddeo, L. Multi-model and Multi-level Development of User Interfaces, Chapter 10, in Seffah, A. & Javahery, H. (Eds.), “Multiple User Interfaces - Cross-Platform Applications and Context-Aware Interfaces”. John Wiley & Sons, New York, November 2003, pp. 193-216.

[25] Vanderdonckt, J., Coutaz, J., Calvary, G., Stanciulescu, A. Multimodality for Plastic User Interfaces: Models, Methods, and Prin-

ciples, Chapter 4, in “Multimodal user interfaces: signals and communication technology”, D. Tzovaras (ed.), Lecture Notes in Electrical Engineering, Springer-Verlag, Berlin, 2007, pp. 61-84.

[26] Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction RO-CHI’2008 (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (eds.). Matrix ROM, Bucarest, 2008, pp. 1–10.

[27] Van Roy, P. and Haridi, S. Concepts, Techniques, and Models of Computer Programming. MIT Press, Cambridge, 2004.

[28] Vandervelpen, Ch., Vanderhulst, G., Luyten, K., and Coninx, K. Light-Weight Distributed Web Interfaces: Preparing the Web for Heterogeneous Environments. Proc. of Int. Conf. on Web Engineering ICWE'2005. Lecture Notes in Computer Science, Vol. 3579. Springer-Verlag, Berlin, 2005, pp. 197-202.