The 4C Reference Model for Distributed User Interfaces

Alexandre Demeure[1], Jean-Sébastien Sottet[1], Gaëlle Calvary[1], Joëlle Coutaz[1],
Vincent Ganneau[1], Jean Vanderdonckt[2]
[1]*LIG, Université de Grenoble, BP53 – F-38041 Grenoble Cedex 9 (France)*
{alexandre.demeure, gaelle.calvary, joelle.coutaz, vincent.ganneau, jean-sebastien.sottet}@imag.fr
[2]*Belgian Lab. of Computer-Human Interaction (BCHI),*
Louvain School of Management (IAG), Université catholique de Louvain,
Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
Jean.vanderdonckt@uclouvain.be

Abstract

Distributed User Interfaces (DUIs) are those interfaces whose different parts can be distributed in time and space on different monitors, screens, and computing platforms, depending on several parameters expressing the context of use, such as the user, the computing platform, and the physical environment in which the user is carrying out her interactive task. To understand and classify existing approaches for DUIs and to identify underexplored situations of DUIs, a reference model for DUIs is introduced that examines DUIs according to four 'C' dimensions: computation (what is distributed?), communication (when is it distributed?), coordination (who is it distributed?), and configuration (from where and to where is the distribution operated?). At the core of this reference model exists the original notion of user interface habitat, that is the place where a particular type of user interface is normally found. According to this notion, it is possible to explore and investigate a wide spectrum of DUIs among which we exemplify several cases coming from our existing research and development of DUIs.

1. Introduction

With the advent of ubiquitous computing and the ever increasing amount of computing platforms, the user is confronted with more and more situations where she is invited to move from one platform to another while carrying out her interactive task, across several platforms or even with the platforms themselves. The ultimate situation is when the user is carrying out a task in a physical environment where several systems are concurrently working on different physical platforms, but forming a single task oriented user interface from the user's viewpoint. All these situations represent typical cases of Distributed User Interfaces (DUIs) where one or many parts or whole of one or many user interfaces are distributed in time and space depending on several parameters of the context of use, such as the user, the computing platform, and the physical environment where the task is carried out [5]. For instance, in a medical environment, a surgeon may be confronted to several screens coming from one or many platforms that are distributed in space, but without any migration across them. Today, there is an ontological confusion between the various terms borrowed to refer to various cases of DUIs such as: migration [14], migratory UIs [1,2], migratable UIs [9], grouping/ungrouping [7], portable [10], tunable [13], mutable [13], transformable [9], reconfigurable [8], retargetable [4], composition/ decomposition [20]. In addition to that confusion, a need exists for consolidating this very recent branch of research dedicated to DUIs. One the one hand, several DUI systems exist such as IAM [6], i-Land [17], Stanford Interactive Mural [11], Aura [16], ConnecTables [18,19], Dygimes [20], DistriXML [9]. So far, no state of the art of these systems has been conducted and no reference framework has been proposed to better perceive the differences existing between these terms, referring to various situations. For each term, different implementations have been realized that are often located at different levels of abstraction and different parts of the UI. For instance, the migration of a User Interface (UI) may be interpreted as the action of transferring a UI from a computing platform to another one, such as from a desktop computer to a handheld device. A UI is said to be migratable if it has the migration ability. If we stick to this definition, several ways have been implemented to achieve migratability: X11 remote displays (www.x.org/), Virtual Network Computing (VNC - www.uk.research.att.com/vnc/), Windows Terminal Server (http://www.microsoft.com/windows2000/technologies/terminal/default.asp) all allow migrating a window from one screen to another at the window manager level. But this is not the same level

0-7695-3093-1/08 $25.00 © 2008 IEEE
DOI 10.1109/ICAS.2008.34

as application migration across workstations [2] or task-oriented migration of parts or whole of the UI [1]. Partitioning a window across several screens, at the physical or at the logical level, is not comparable and involve different systems of coordinates (Fig. 1) until reaching the definition of interaction surfaces [7].

For each DUI term, we need to differentiate the different implementations possible by locating them on a reference frame-work. Such a framework would also be fundamental to understand the human factors posed by DUIs: when a surgeon must collect information from several screens simultaneously, a significant increase of the cognitive work load may arise. Several usability problems are raised by distributing UIs across several screens such as peripheral vision difficulties [9], risk of cognitive overload [12], configuration difficulties [12], and differences of perception [11].

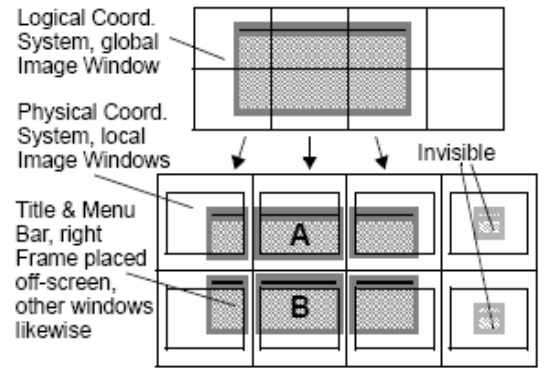


Fig. 1. Partitioning of a window across several screens to obtain a distributed user interface.

For all these reasons and other, and given the fact no other framework allows to represent platforms, spatial relationship and digital elements in the same time, we assigned ourselves the goal of defining a reference framework for DUIs that would allow us to clearly distinguish existing terms, precisely locate related work according to these terms, and identify underexplored areas. For this purpose, the remainder of this paper is structured as follows: Section 2 defines the UI habitat, the notion central to this framework that will allow term differentiation; then, four 'C' of the framework will represent four classification dimensions on how to understand DUIs in Section 3. Section 4 will then illustrate the 4C reference framework for DUIs by expressing several examples coming from our research and Section 5 will report on the benefits of this framework.

2. The notion of habitat as a central concept for distribution

In the real world, a habitat is referred to as the typical place where an animal or a vegetal can be found. By analogy, the *habitat* of an interactive system consists of the configuration of the interactive system. By definition, a habitat is a graph, constituted of nodes (e.g., CPUs, FCs, UIs, Actuators) and links betweens those nodes. We describe here some of these nodes:

- *Logical Space:* It defines a coordinates system.
- *Zone:* It defines an area of a logical space.
- *CPU:* The Central Processing Unit (CPU) computes the interactive system. A CPU can be seen from the programmer point of view as a single resource even if several physical processors are used. It defines its own logical space.
- *Actuator:* It is a physical space's zone making perceivable digital user interface entities. It is managed by a CPU. An actuator can be seen as a single entity from the programmer's point of view (e.g., a wall screen, two loud speakers). An actuator makes perceivable a part of several coordinates systems. A screen makes perceivable a part of windows.
- *Functional Core:* It makes available a set of services. A functional core (FC) can be connected to other FCs. It is managed by CPUs.
- *User Interface:* It's a digital entity defining its own coordinate system. It can be directly or indirectly mapped on an actuator (e.g. a window is define in the logical space of the desktop, which is mapped on a screen).

Nodes of a habitat are described in the UML diagram of Fig. 2 where relations between nodes will be used in section 4.

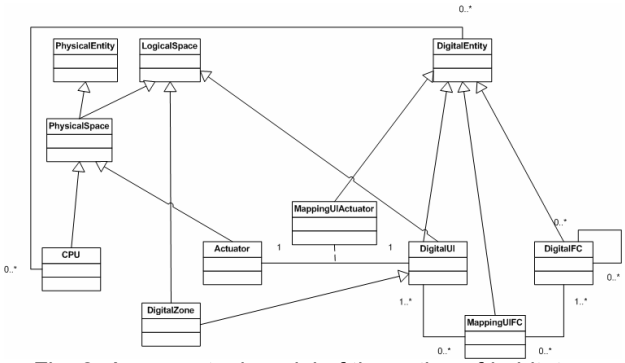


Fig. 2. A conceptual model of the notion of habitat.

Error! Reference source not found. illustrates different kinds of habitat, regardless to the functional core. Each case is illustrated by a typical interactive system using it. For instance, a laptop connected to a dual screen illustrate the <1 CPU, 1 digital space, 2 screens> case. Thanks to the graph structure of a habitat, it becomes possible to formally describe reference terms for DUIs. All terms we will define are about UI nodes. In a first time, we describe terms involving only one habitat. Let us introduce some utilities functions:

- **Rel(i, j) :** *i* and *j* are nodes, there is a relation *Rel* from *i* to *j*.

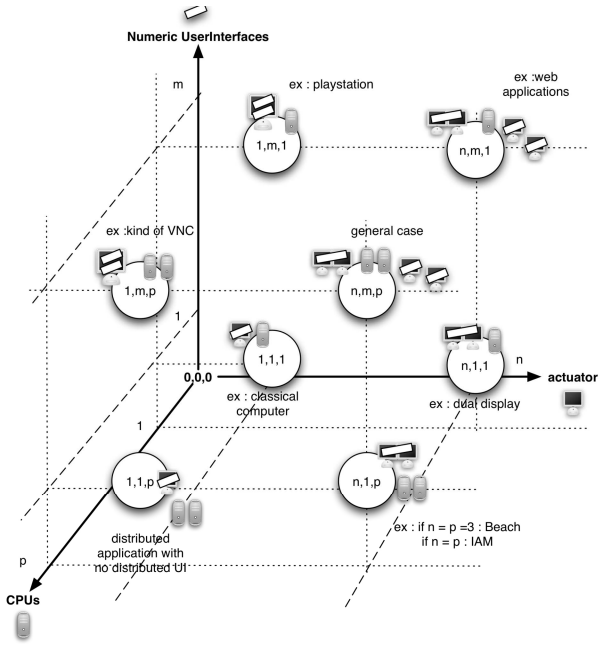


Fig. 3. Different <#CPU, #DigitalSpace, #Actuators> habitats.

- **Replicas(i, j) :** *i* and *j* are nodes of a same type, that is they are considered as replicas.
- **Way :** it is a path in a habitat graph.
- **Mothers(I : DigitalEntity; X:TYPE; R:Relation)** : {m:X | Rel(m, I)•m}

Let us introduce function involving only one habitat.

- **LOCATIONS :** it is a set of triplet <CPU, sm : set of MappingUIA, w : way of DigitalEntity >
- **Locations_where_I_is_displayable(L : LOCATIONS) :** the subset of L where CPU are managing Effectors mapped on a node {h:H | h.sm≠∅ • h}
- **AreXScattered(si : set of DigitalUI) :** Define whether a set of DigitalUI is scattered from a X point of view. X can be CPUs or Actuators. Actually, scattering property can be seen as distribution property. This is from a system point of view. Indeed, with this definition a DIgitalUI can be scattered among a set of screen even if it is only displayed on one of them. (This function will indicate on which screen the DigitalUI is *displayable*).
- **SSX** : set of sets of X | SSX={i:si • getX(i)} • α
  - *strong condition* : Two different DigitalUI have no common x∈X such as α ≡ (¬∃sx1,sx2 : SSX | sx1∩sx2≠∅)
  - *weak condition* : At least two DigitalUI have different X such that α ≡ (∃sx1,sx2 : SSX | sx1≠sx2)
- **HowMuchIsActuated(Z: DigitalZone)** : Define what percentage of a zone is displayed considering actuators where it is displayable. L = Locations(Z), sz = ⋃{l:L • {mapping:l.sm • Zone(Z.zone, l.way, mapping)} ← Size(⋃(sz)) / Size(Z.zone)
- **Zone(z:Zone, w:Way, m:MappingUIA)** : For a zone z, define what part of it is actually displayed by an actuator, given the mapping from Actuator coordinate to DigitalZone coordinate (f_phy_to_Digital function)
  w = w_root→i
  (m.i=i)⇒(ZoneActuatorInInfoRef=m.f_phy_to_Digital (m.actuator.zone) ← ZoneActuatorInInfoRef ∩z)
  (m.i≠i) ⇒(w=w1→m.i→w2→I ←Zone(i.Transfo_Zone(zone), w_root, m))

Let us illustrate a function involving two habitats:

- **Migrations(I : DigitalUI, H1, H2 : Habitat) :** A doublet <Q,A> where Q and A are LOCATIONS. Q represents LOCATIONS leaved by 'I' between H1 and H2 and A LOCATIONS where 'I' arrived.
  L1:H1.Habitat(I), L2:H2.Habitat(I)
  <{l1:L1 | l1=<cpu1, sm1, w1> ∧ (¬∃l2:L2 | l2=<cpu2,sm2,w2>∧cp1=cpu2∧w1=w2∧sm1⊆sm2)• l1}, {h2:H2 | h2=<cpu2, sm2, w2> ∧ (¬∃l1:L1 | l1=<cpu1,sm1,w1>∧cp1=cpu2∧w1=w2∧sm2⊆sm1)• l2}>

Actually, a lot of properties can be expressed using the notion of habitat. Various authors have a perhaps different definition of what a DUI could be or how the distribution of UI elements could be operated. Therefore, we would like to provide a formal definition. For example, the notion of distribution of a UI can be: "My UI is *displayable* on different screens", "My UI is *displayed* on different screens", "Parts of my UI are (strongly, weakly) scattered among screens", "my UI is decomposed into…". Using the habitat notion, all of those properties can be expressed. The general properties of UI habitat can serve as a common ground for expressing various manifestations of DUIs, which are developed in the next section throughout the four dimensions of the reference framework.

3. The four dimensions of the 4C Reference Framework

In this section, we define the four 'C' representing the reference framework for DUIs.

3.1 C1: Computation

C1 represents the computation of a DUI, i.e. **what** are the elements to be distributed during the operation. Theoretically speaking, all components of the interactive application could be distributed, but we are here mainly interested by the UI. Based on the ARCH meta-

model, we envision distributing the following layers: presentation, logical presentation, control, functional code and its adapter. Presentation is materialized through a Concrete UI (CUI) as in the Cameleon reference framework [3], while logical presentation is related to the Abstract UI (AUI). Therefore, the part that is subject to distribution could be any part or whole of the CUI. The CUI model decomposes any UI into Concrete Interaction Object (CIOs) that are characterized by various attributes (*id*, *name*, *icon*, *defaultContent*, *defaultValue*) and sub-typed into one of the two categories: *graphicalContainer* for all widgets containing other widgets such as page, window, frame, dialog box, table, or *graphicalIndividualComponent* for all other traditional widgets: text, video, image, radio button, drawing canvas,….For instance, the whole UI, some windows and dialog boxes, some selection of components of these windows and dialog boxes, or individual components. For this purpose, we need to define the notion of **splittability**: a graphical container is said to be *splittable*, respectively *unsplittable*, if all their graphical individual components could, respectively could not, be presented separately depending on the constraints imposed by the user's task corresponding to the container. With respect to the habitat notion, splittability is defined as the fact whether the logical reference of each element in a container has to be *C.ref* where *C* is a logical reference of the container. In fact, splittability is the ability for any part of an interactive system to be ungrouped. For this purpose, an attribute i*sSpittable* is assigned to any container. The UI portion subject to distribution is therefore any node in the CUI hierarchy or combination of nodes or leaf nodes.

3.2 C2: Communication

C2 represents the dimension of distribution time, i.e. **when** are the elements of an interactive system distributed. For this purpose, we need to define a time line that segregates the distribution operation whether it is executed at design-time or at execution-time or between. Based on the taxonomy of adaptive middleware [12] and classical definition of process migration [13], the UI distribution is said to be *static* or *dynamic* depending on the distribution time: static distribution methods take place at development, compile, or load time, whereas dynamic distribution refers to methods that can be applied at runtime (Fig. 4). If an interactive system is scheduled for distribution at development time, then we consider that the definition of the habitats is hardwired into the interactive system and cannot be changed without recoding. Alternatively, a developer or user can schedule a limited set of distribution capabilities at compile time or link time by configuring the interactive system for a particular set of predefined habitats. A UI is said to be *customizable* when the interactive system only requires recompilation or relinking to fit to a new set of possible habitats that are supported. A *configurable* UI delays the final decision on the logical CPUs to use in the current environment until a running UI loads the corresponding component. Load-time composition can be considered as a type of static distribution, but it is more dynamic than other static cases. When the UI requests the loading of a new component, the control might select from a list of UIs with different capabilities of supporting various habitats, choosing the one that most closely matches the new habitat. For example, if a user starts a UI on a handheld computer, the runtime system might load a minimal display component to guarantee proper presentation. Other load-time approaches work by dynamically modifying the class itself as it is loaded. In *dynamic distribution*, two types of approaches can be distinguished according to whether the distributor can modify the UI itself. A *tunable* UI is a UI that cannot be submitted to any change of underlying code, but that can be tuned according to user's needs that are known only at run-time. A *mutable* UI is a UI that can be submitted to a change of its underlying code, e.g., by decomposition and recomposition of components.

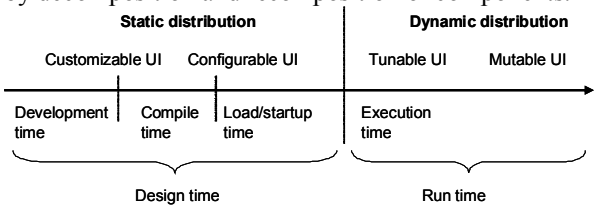


Fig. 4. Time of distribution.

3.3 C3: Coordination

C3 represents the dimension of distribution responsibility, i.e. **who** is distributing the interactive system. To support all these different cases of distribution, a special UI is required that will perform the required steps to conduct the distribution, such as identification of distribution possibilities, proposal for distributions, selection of an appropriate alternative, and execution of the distribution itself. Since these types of distributions and underlying steps require complex handling of UI events and procedures, the UI responsible for distribution is even more complex and not always visible to the eyes of the end user. This UI is referred to as the *meta-user interface*, i.e. the UI for controlling the UI distribution of the interactive systems. In most of research/development projects involving some form of migration, the meta-UI is implemented in very different ways with different manifestations. It is not made explicit whether the meta-UI is *system initiated* (the system initiates the distribution), *user-initiated* (the user initiates the distribution), or *mixed-initiated* (the user and the system collaborate to perform the distribu-

tion together). In some special cases, the distribution could be ensured by a *third-party*, an agent external to the user and the system which is responsible for conducting the distribution. The above parameters could be applied separately and individually to all operations involved in the distribution:

- *Detection*: which agent detects the need for distribution and according to which parameters from the context of use?
- *Computation*: which agent computes the different alternatives for conducting the distribution? Regarding this aspect, a typology of distribution operations could be imagined such as, but not limited to: (un)grouping, merging,...
- *Selection*: which agent selects the most appropriate alternative among the computed ones?
- *Execution*: which agent actually executes the alternative selected for the distribution?

For example, a user may decide that there is a need to do so and selects various portions of the UI which could then be migrated to other platforms she is using. When the task is finished, she may want to recall all migrated portions to restore the initial UI. In this case, detection, computation, and selection are user-initiated while execution is system-initiated.

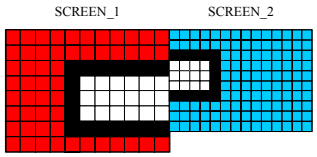
3.4 C4: Configuration

C4 represents **where** a UI is distributed. It therefore poses the problem of how to distribute it. Should we keep the same representation, adapt it or change it? Indeed, a representation adapted for a location l1 may not be adapted for a location l2. We can illustrate this considering the particular GUI case. Every single part of a GUI can be seen as a bloc of pixels. Particular problems arise when a part of the UI go from one screen to another. It may pose the problem of how to represent the UI on this new screen. We identified three ways to achieve this:

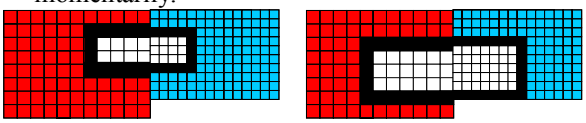
1. Conserving *physical pixels*: By physical pixel we mean the smallest point we can address on the video card and/or on the screen. Conserving physical pixels mean that a bloc of L x H physical pixels in the location l1 would be represented by the same bloc of L x H physical pixel in the location l2.
2. Conserving *logical pixels*: By logical pixel we mean a point independent of the video card and the screen. A logical pixel will be rendered as a bloc of pixels (1 logical pixel ⇔ L x H physical pixels, L, H ∈ $\Re$+). Conserving logical pixels mean that a bloc of L x H logical pixels in the location l1 would be represented by the same bloc of L x H logical pixel in the location l2. Logical pixels from l1 and l2 could have different size when rendered to physical pixels.
3. Conserving *semantic pixels*: By semantic pixels we mean a set of pixel expressing a concept. Conserving semantic pixels means conserving the same meaning for two blocs of pixels on different locations, e.g., a list box and a drop down list both express the concept (task) of choice.

The adequacy of those three approaches wrt. the context of use will depend on what we want to preserve between locations. In particular, if we want to conserve rendering size (i.e. size of the UI on the screen), some problems may occur. Three cases can be distinguished:

- Conserving *physical pixels* can cause some trouble, especially if we want to obtain a spatial continuity when distributing UI across several screens. This is typically the example of IAM. Problem will then appear if the size of physical pixels in l1 is different from the one of l2, conserving physical pixels will then lead to a different size of GUI and may result in alignment problems. Moreover, if screens have different resolutions, it may be impossible to fully render the GUI on some of them.



- Conserving *logical pixels* aim to solve some the alignment problem. However it can't fully solve it in every case. Indeed, target physical pixels size could be too big, witch would result in lost of information (1mm⇔1 pixels on source while 1mm⇔0,8 pixels on destination for example). In fact, to be sure to not loose any information, pixel size of target should be half the size of source one (Shannon's theorem) OR if we consider enough small pixels, pixels should have the same size. We can consider pixels alignment as a minor problem momentarily.



- In fact, logical pixels adequacy is a bit more complex. It depends on the logical pixels size on the surfaces. Let's consider that a logical pixel on surface take LxH physical pixels. The adequacy of logical pixel is: max(L/2,1)* max(H/2,1). The adequacy does not only depend on the relative physical pixels size but also on the logical pixel size: e.g., if we want to full screen display a 100x100 pixels image on a 1024x768 and on a 800x600 screens, adequacy will be optimal on both, despite that physical pixel sizes do not respect Shannon's theorem.
- Conserving *semantic pixels*: it can lead to a great discontinuity. To minimise it notions such as graceful degradation techniques should be applied. For

instance, when a UI is reduced from a classis desktop to a more constrained platform such as a TabletPC, not only the UI should be degraded, but this degradation should preserve the usability of the resulting UI for continuity.

We propose a 3D space to describe different configuration we can meet when displaying logical pixels on physical ones. Let's consider a GUI S, it take S.L x S.H pixels. Let's consider we want to migrate it on a surface D witch is D.L x D.H pixels. Let's also consider that pixel size on S is S.Pix and pixel size on D is D.Pix. Fig. 5 illustrates possibilities in the case D.Pix / S.Pix > 1. The more the case is in the top right corner, the better it is. We could have an evaluation function to determine the pertinence of each adaptation type:

- Physical pixel conservation: PPC (R_L, R_H, R_P) = max(R_L, 1) * max(R_H, 1)
- Logical pixel conservation : LPC(R_L, R_H, R_P) = max (R_L* R_P, 1)*max(R_H* R_P, 1)*max(1 / (2*R_P), 1). If we consider pixel small enough or if we do not matter about the pixel alignment between S and D then we have the particular case : LPC (R_L, R_H, 1) = PPC (R_L, R_H, 1)
- Semantical pixel conservation: Is welcome when others conservation have low level of adequacy.

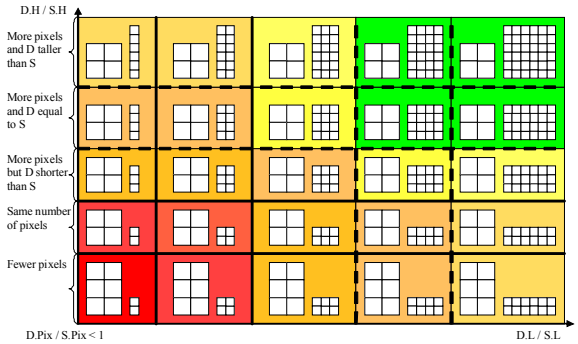


Fig. 5. Relative dimensions when destination pixel size is lower than source one.

4 Instantiations of the 4C Reference Framework

In order to improve readability in the rest of the document, Fig. introduces the notation used in the rest.

A — B A and B are CPU. They can communicate.
A —● B : A manage B. A is a CPU, B is an Actuator or an InfoNode or a Mapping
A —> B : B is defined with respect to A. A and B inherit from T.
A —▶ B : A and B are functional cores. B depend on A.
A - - - B : A and B are conceptually the same. They are replicas.
M ---- B : M define a mapping between A and B. A is an Actuator and B an InfoNode.

Fig. 6. Relations between nodes of a habitat graph.

4.1 Instantiation #1: Extending a Desktop with another Screen or Monitor

In this case, only the rendering zone is extended using another screen instance, here called "SCREEN 2" (Fig. 7). When the two screens are kept separate, a visual discontinuity is induced, which may decrease when they are joined, but will never totally disappear due to the rupture caused by the bezels.

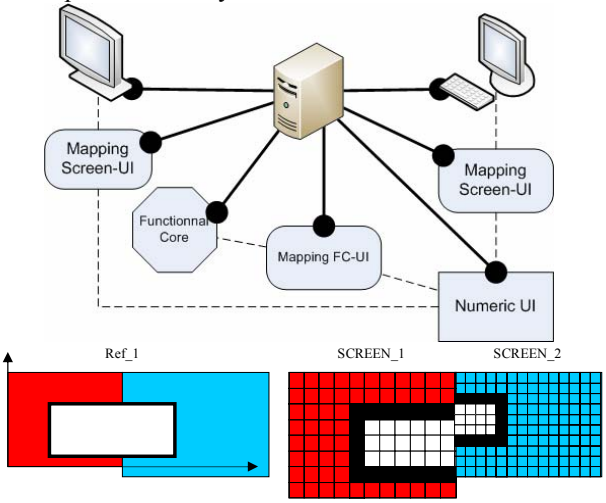
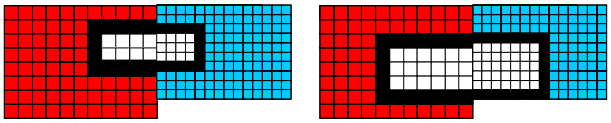


Fig. 7. At physical pixel level, spatial discontinuity can often be observed.

4.2 Instantiation #2: The IAM environment

The primary principle of IAM is to replicate logical spaces between elementary platforms. Being aware of the spatial discontinuity problem, an environment like IAM adapts logical space to prevent it. Logical space is then stretched, rotated and/or translated. But if we consider previous case, we can have two type of solution depending on where the UI come from (SCREEN_2 (left case) or SCREEN_1 (right case)).



The left case above shows that some information will be lost on SCREEN_1, due to its big pixel size. Similarly, the right case could also cause some loss of information if pixel size from SCREEN_2 is not half of the SCREEN_1's pixel size. Same problems may occur when using anti-aliasing: a pixel from SCREEN_1 should not necessarily correspond a only pixel from SCREEN_2 and reciprocally (Fig. 8). Let us illustrate how the UI is replicated in the top right case (Fig. 9).

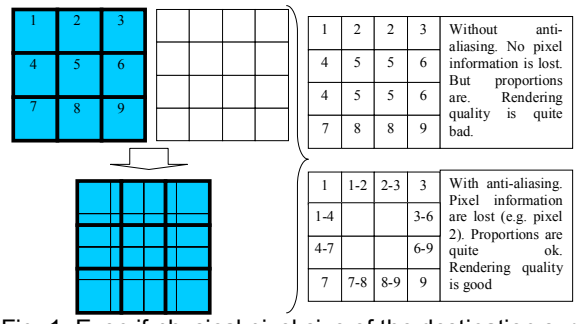


Fig. 1. Even if physical pixel size of the destination surface is smaller, problems can occur.

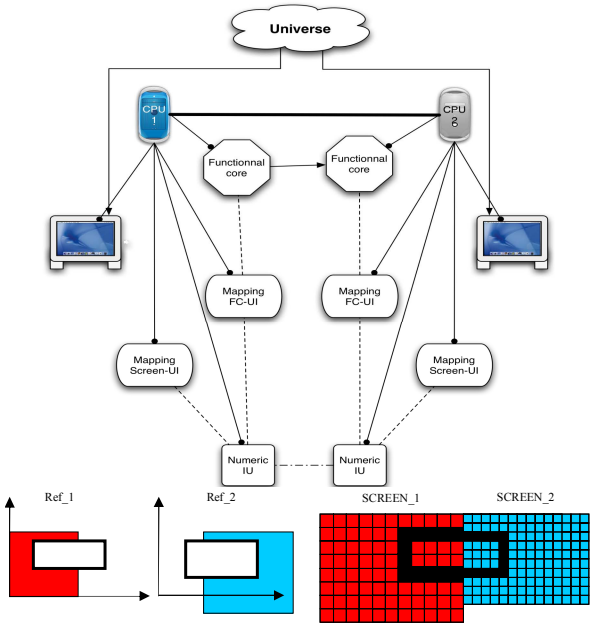


Fig. 2. CPU1 is the server where the UI is created. UI is replicated among IAM clients. If a client drops down, the server does create another UI.

4.3 Instantiation #3: The Sedan-Bouillon Tourist Application with Plasticity

"Sedan-Bouillon" is a web site that aims at promoting tourism in the regions of Sedan and Bouillon in France and Belgium (www.bouillon-sedan.com/). It targets tourists whose mother tongue is French, German, and Dutch. It provides tourists with four kinds of information or services: general information about the region, guiding information for visiting the region (a pre-selection of specific monuments of interest), logistic information for sojourning in the region (a preselection of hotels, camping, bred and breakfast, restaurants), and access to a set of relevant documentation. Initially, the web site has been designed for PC screens only. A plastic light weight version (called LSB for Light Sedan-Bouillon) has been developed in for exploring the distribution of a web site across a PC screen and a PDA. LSB is limited to French speaking users and covers the hotel browsing task only. But LSB is plastic since it promotes a user-initiated distribution of the UI portions among the web browsers with which the user is connected to the web site. LSB is composed of three workspaces (Fig. 10): a title, a navigation menu and informational content. The navigation menu is augmented with a "meta-UI" link that allows the user to control the UI distribution across the resources of the interactive space: for each workspace, the user can specify the platform on which it can be displayed. To comfortably browse the site from the sofa, the user turns on a PDA and connects to the web site with the same identifier. A meta UI appears that informs the user that she is currently using two browsers, and that she can redistribute the user interface across the resources available in the interactive space: the three workspaces are mentioned and the two browsers are identified. The, the user can specify which workspace will run on which platform. In our case, the user asks for the title and the contents to be displayed on the PC, while keeping the title (titre) and the navigation menu (navigation) to be rendered on the PDA. The PC and the PDA screens are then updated accordingly. Fig. 11 characterizes this situation the UI workspaces could be distributed via the meta-UI, either individually or together provided that they included.

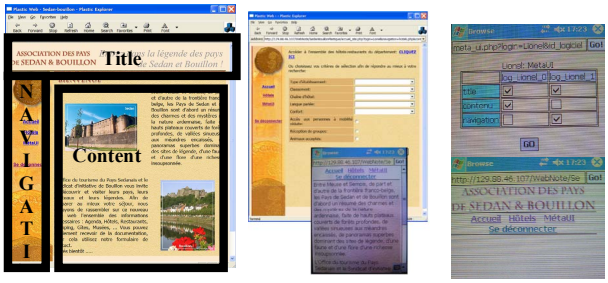


Fig. 3. The Sedan-Bouillon application.

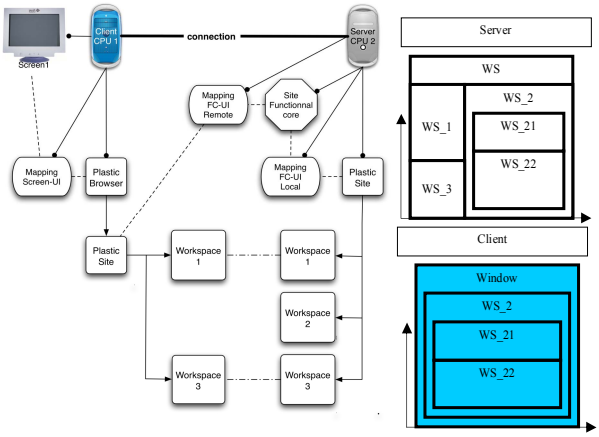


Fig. 4. FC is on the server side. Only UI is replicated. If the client server connexion drops down, UI remain on client side but cannot drive the FC anymore.

4.4 Instantiation #4: CamNote

The CamNote application is a PowerPoint-like application where the slide controller could be run on a PC hosting the slides or on a PDA to serve as a remote controller. The PC Remote Controler, however, does not run on the PDA since its capabilities are not equivalent, in particular in terms of runnable applications. Therefore, it has to be substituted by another one which is tailored to the PDA.

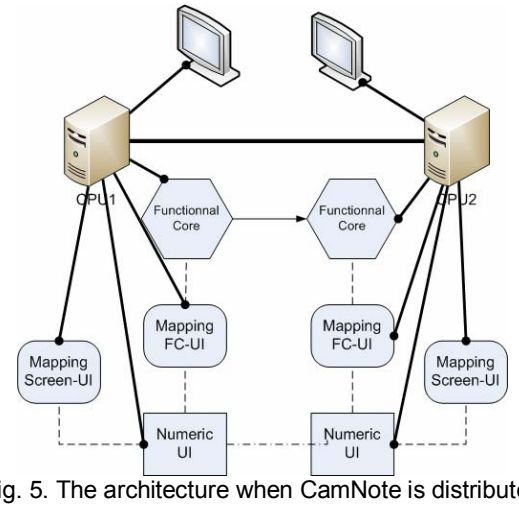
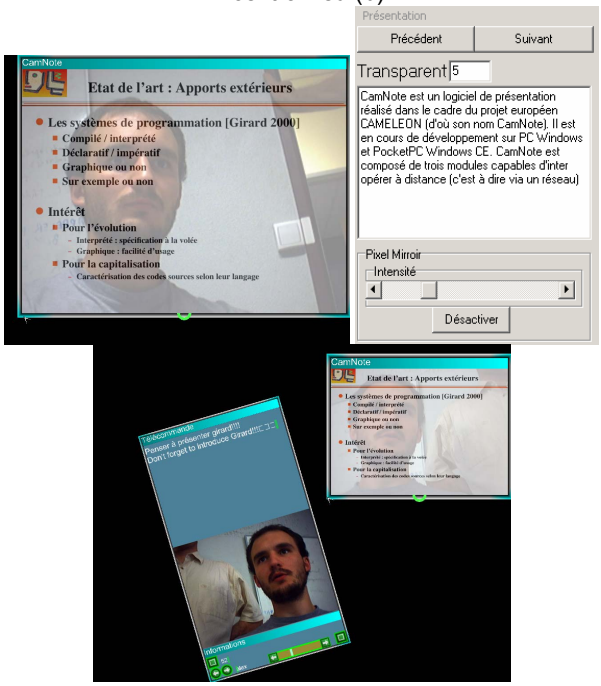


Fig. 5. The architecture when CamNote is distributed across the PC and the PDA (a) and when it is re-centralized (b).



The above figure depicts the whole system when distributed between a PC and a PDA, a situation whose architecture is reproduced in Fig. 12. Only functional core is distributed. Indeed, native UI descriptions on a PC and on a PDA are quite different. Using a same toolkit would have been inappropriate. When the PDA disappears, the remote controller function is migrated to the PC. A DigitalUI node is then created and takes place with respect to the DigitalUI root node of the CamNote interactive system. When then system is re-centralized (this architecture is depicted in), the window containing the remote controller could be merged with the slide show or separated. When this operation occurs, the window is being rotated as represented in the figure below in order to animate the transitions between the two situations: centralized (Fig. 13) and decentralized.

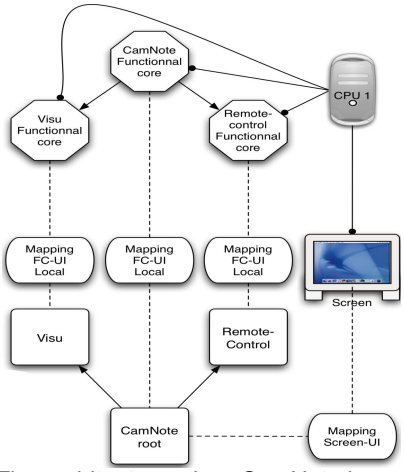


Fig. 13. The architecture when CamNote is centralized.

5 Conclusion

In this paper, we have motivated, introduced, defined and exemplified a reference framework for distributed user interfaces (DUIs) that is explicitly based on criteria defined along four dimensions: computation (what is distributed?), communication (how is it distributed?), coordination (when is it distributed?), and configuration (from where and to where is the distribution operated?). Thanks to this combination, it is possible to characterize a wide array of DUIs. At first glance, the most typical cases of DUI involve the following situations which could be easily characterized on the framework defined in Fig. 2:

- The *multiple monitors* situation: a single user may connect a dual screen on her main computer so as to extend the display area. In this case, there is only one CPU, one DigitalUI, but several actuators.
- The *multi-platform* situation: a single user may use simultaneously two computing platforms (hence, two different CPUs) which are interconnected, thus exhibiting the capability of rendering a portion of each other's UI by exchanging them.
- The *detachable user interface* situation: this is particular case of the multi-platform situation where one platform may delegate (detaching) por-

tions of its UI to a secondary platform (attaching). When the task is finished, the detached portion could be restored to its initial position (re-attaching).

- The *fully distributable* situation: this is a situation where each platform could operate a distribution of its workspaces to other platforms at run-time. One platform could for instance replicate a portion of its UI to another platform (e.g., a shared one) and receive another portion of another UI coming from another platform.

The framework defined in this paper does not take into account the dimension of collaboration between or across users. This dimension is a natural extension: for instance, a DUI could take place not only for a single user (as we addressed in this paper) or across several users (which is not addressed here).

6. Acknowledgments

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).

7. References

[1] Bandelloni, R. and Paternò, F., "Flexible Interface Migration", Proceedings of ACM Con. On Intelligent User Interfaces IUI'04 (Funchal, January 2004), ACM Press, New York, 2004, pp. 148–155.

[2] Bharat, K.A. and Cardelli, L., "Migratory Applications Distributed User Interfaces", Proc. of ACM Conf. on User Interface Software Technology UIST'95, ACM Press, New York, 1995, pp. 133–142.

[3] Bouillon, L. and Vanderdonckt, J., "Retargeting Web Pages to other Computing Platforms with VAQUITA", Proc. of IEEE Working Conf. on Reverse Engineering WCRE'2002 (Richmond, 28 October-1 November 2002), IEEE Comp. Society Press, 2002, pp. 339–348.

[4] 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*, 15(3), 2003, pp. 289–308.

[5] Coulouris, G., Dollimore, J., and Kindberg, T., *Distributed Systems: Concepts and Design*, 3rd edition. Addison-Wesley, Reading, 2001.

[6] Coutaz, J., Balme, L., Lachenal, Ch., and Barralon, N., "Software Infrastructure for Distributed Migratable User Interfaces", Proc. of UbiHCISys Workshop on UbiComp 2003, 2003.

[7] Coutaz, J., Lachenal, Ch., and Dupuy-Chessa, S., "Ontology for Multi-Surface Interaction", Proc. of IFIP Conf. on Human-Computer Interaction Interact'2003, IOS Press, Amsterdam, 2003, pp. 447–454.

[8] Grolaux, D., Vanderdonckt, J., and Van Roy, P., "Attach me, Detach me, Assemble me like You Work", Proc. of 10th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2005 (Rome, 12-16 September 2005), Lecture Notes in Computer Science, Vol. 3585, Springer, Berlin, 2005, pp. 198-212.

[9] Grolaux, D., Van Roy, P., and Vanderdonckt, J., "Migratable User Interfaces: Beyond Migratory User Interfaces", Proc. of 1st IEEE-ACM Annual International Conference 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.

[10] Grudin, J., "Partitioning Digital Worlds: Focal and Peripheral Awareness in Multiple Monitor Use", Proc. of Conf. on Human Factors in Computing Systems CHI'2001, ACM Press, New York, 2001, pp. 458–465.

[11] Guimbretière, F., Stone, M., and Winograd, T., "Fluid Interaction with High-resolution Wall-size Displays", Proc. of ACM Conf. on User Interface Software Technology UIST'2001, ACM Press, New York, 2001.

[12] Jacobson, J., "Configuring Multiscreen Displays with Existing Computer Equipment", Proc. of HFES'2002, HFES, Santa Monica, 2002.

[13] McKinley, P.K., Masoud Sadjadi, S., Kasten, E.P., and Cheng, B.H.C., Composing Adaptive Software, *IEEE Computer*, July 2004, pp. 56–64.

[14] Milojicic, D.S., Douglis, F., Paindaveine, Y., Wheeler, R., and Zhou, S., Process Migration, *ACM Computing Surveys*, 32(3), September 2000, pp. 241–299.

[15] Rekimoto, J., Ullmer, B., and Oba, H., "DataTiles: A Modular Platform for Mixed Physical and Graphical Interactions", Proc. of ACM Conf. on Human Factors in Computing Systems CHI'2001 (Seattle, April 2001), ACM Press, New York, 2001.

[16] Sousa, J. and Garlan, D., "Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments", Proc. of IEEE-IFIP Conf. on Software Architecture, 2002.

[17] Streitz, N. *et al*., i-LAND: An interactive Landscape for Creativity and Innovation, Proc. of ACM Conf. on Human Factors in Computing Systems CHI'99, ACM Press, New York, 1999, pp. 120–127.

[18] Tandler, P.: Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices. In: Proc. of Conf. on Ubiquitous Computing UbiComp'2001, Lecture Notes in Computer Science, Vol. 2201, Springer-Verlag, Berlin, 2001, pp. 96–115.

[19] Tandler, P. Prante, Th., Müller-Tomfelde, Th., Streitz, N., and Steinmetz, R., ConnecTables: Dynamic coupling of displays for the flexible creation of shared workspaces, Proc. of 14th ACM Symp. on UI Software and Tech. UIST'01, ACM Press, New York, 2001, pp. 11–20.

[20] Vandervelpen, Ch. and Coninx, K., "Towards Model-Based Design Support for Distributed User Interfaces", Proc. of NordiCHI'2004, ACM Press, New York, 2004.