Software Infrastructure for Distributed Migratable User Interfaces

Joëlle Coutaz, Lionel Balme, Christophe Lachenal, Nicolas Barralon
CLIPS-IMAG, Univ. Joseph Fourier, BP 53
38041 Grenoble Cedex 9, France
+33 (0)4 76 51 48 54
{Joelle.Coutaz,Lionel.Balme,Christophe.Lachenal,Nicolas.Barralon}@imag.fr
http://www.iihm.fr

ABSTRACT
Based on our own experience in developing software environments for plastic UIs, we argue for a new middleware software infrastructure that extends the functional capabilities of our current windowing systems and toolkits.

Keywords
Distributed user interface, migratable user interface, plastic user interface, software infrastructure, ubiquitous computing.

INTRODUCTION
In HCI, methods and tools have been devised to increase the usability of interactive systems while reducing their development costs (see for example, the model-based approach to the development of user interfaces as well as rapid prototyping tools). Although supposed to be general, the current foundational tools, e.g., windowing systems and toolkits, make a number of implicit assumptions that simplify implementation issues and favour system performance. In the context of ubiquitous computing, however, the assumptions made in windowing systems turn out to be limiting factors to "get the UI out of the box" so that users can interact with the environment using a dynamic set of interaction resources[1] that can be opportunistically composed, borrowed and lent [5].

The state of the art in ubiquitous computing shows early examples of interactive systems, motivated by human-centered concerns, that are based on the dynamic composition of interaction resources: Distributed UI as in Rekimoto's Pick and Drop [17], dynamic docking of multiple displays to enlarge real screen estate (e.g., the ConnecTable or Hinckley's work), Migratable UI as in I-land [19] and Seescoa [11], or the Personal Server approach that promotes the borrowing of near-by interaction resources [22], etc. As far as we know, all of these prototypes, except perhaps I-land, have been developed as concept demonstrators reusing and hacking the current foundational tools of the GUI technology.

[1] By interaction resources, we mean any physical object that can serve as a surface to display information content (e.g., an augmented table), as an instrument to manipulate informational content (e.g., fingers, pens and phicons), or both at the same time (the surface of a finger can be used to project information on it).

If the scientific community aims at supporting a sound approach to the development of distributed migratable UI, we need to revise our foundational tools, which, by definition, set the basis for the development of higher level of abstraction tools. In turn, experience shows that these high level tools increase the chance to develop usable user interfaces .

For example, in our research group, we are currently concerned with the definition of models and tools that support the design and development of plastic UIs, e.g., UIs that can dynamically adapt when they migrate to different platforms, or when they are dynamically distributed across a dynamic set of interaction resources [4, 21]. We have been able to devise and develop ARTStudio [4], an environment that supports the specification and generation of Java-based plastic centralized UIs that can adapt to PDA, Cell phones and workstations (See the Cameleon project for additional results along with the Teresa [14] and Vaquita tools). These tools support well-established methods that ensure the usability of the resulting UI. But they do so for centralized UIs where a virtual machine like JVM or a web browser is sufficient for supporting the portability of the generated code. When it comes to address distributed plastic UI over a dynamic set of heterogeneous resources, our specification and generation tools are impeded by the existing underlying software.

In the following section, we present our analysis of the limiting factors that we had to face when we started considering plasticity for distributed migratable UIs. We claim that these limitations need to be addressed by the software community in order to enable the development of HCI-centered development tools. We then outline our proposal for a middleware run-time infrastructure under implementation that addresses these new requirements.

LIMITING FACTORS IN CURRENT FOUNDATIONAL TOOLS

1. *The window model is biased by the workstation screen. Instead, the concept of window must be replaced with that of a physical surface that explicitly models its physical attributes*. Today, windowing systems model windows as rectangular drawables whose borders are constrained to be parallel to that of the display. This model is based on the (wrong) the assumption (for ubiquitous

computing) that users keep facing a vertical screen and that the rendering surface is rectangular. With the proliferation of video projectors, it is increasingly popular to project window contents on horizontal surfaces such as circular tables. In this situation, users should be able to rotate the digital content so that everyone can share information without twisting their neck. Today, rotating windows and user interfaces must be implemented from scratch from low-level graphics primitives. Jazz/PAD++ is a toolkit that offers the basics for implementing rotating, zoomable user interfaces for centralized UIs [1]. Similarly, physical attributes of rendering surfaces, such as size, shape and colour, must be made available to the application program: one does not want to render information on a rectangular white surface in the same way as on a circular table covered with a red cloth. In turn, the capture of the physical attributes of a surface calls for new requirements on the sensing technology (precision, latency, robustness).

2. *Geometrical relationships between physical displays are poorly modelled. Instead, the (possibly 3D) spatial relationships between the interaction resources and the user should be explicitly modelled, dynamically acquired and maintained.* Windowing systems are able to support a limited number of screens. In addition, the relative location of the display screens must be set up by the user through dedicated system forms. As a result, the user is in charge of additional articulatory tasks. Again, because they are screen-centric, windowing systems do not support topologies that include the surrounding environment (e.g., walls, tables, users' location with respect to the display surfaces, etc.). However, for many novel interactive systems, the topology of the rendering surfaces matters: for example, 3D rendering on a vertical surface and 2D presentation on a horizontal surface [3, 15, 17].
3. *Only single instances of input instruments are supported. Instead, the system should be able to support any number of instances of an instrument class.* In current windowing systems, the reference workstation is supposed to have one single mouse and keyboard. On systems like MacOS, it is possible to plug two physical mice. Unfortunately, they are linked to the same interruption level and are modelled by the event manager as a device type, not as a device instance identifier. As a result, multi-user applications such as MMM [2] whose users share the same screen with multiple mice, require the underlying toolkit and event manager to be revisited as in MID [9].
4. *Interaction is confined to the resources of a single workstation. Instead, we need to distribute UI's across a set of interaction resources managed by a cluster of possibly heterogeneous machines.* Applications like I-land [19] and Rekimoto's augmented surfaces [17] require the aggregation of multiple computers. BEACH supports the distribution of user interfaces across a set of homogeneous resources both in term of hardware and operating systems [20]. Unfortunately, ubiquitous computing requires the capacity to handle heterogeneous sets of resources as well. For example, pocket-size computers can play the role of input devices to control information displayed on wall-mounted electronic boards as in Pebbles [12]. In addition, the distribution of the user interface may be static as in Rekimoto's Pick and Drop [16], or distribution may be dynamic as for the Dynawall [19], which then requires additional functionalities such as mechanisms for state recovery.
5. *Absence of dynamic discovery of interaction resources*. *Instead, in ubiquitous computing, interaction resources appear and disappear opportunistically*: users can upload situated information on their private PDA as they pass by a public active wall. Two users who meet serendipitously in the street, may want to start a collaborative activity by bringing together their PDA's to form a larger interactional space. BEACH supports this possibility provided that the resources involved are homogeneous (Cf. the ConnecTables). GaiaOS and Aura [18] aim at supporting heterogeneity and dynamic migration. However, they do not address the problem of developing distributed user interfaces.

In our research group, we are currently working on a number of these issues. The following section presents the flavor of our approach.

THE CAMELEON RUN TIME INFRASTRUCTURE FOR DISTRIBUTED MIGRATABLE PLASTIC UI'S

Figure 1 shows the global functional decomposition of a run-time infrastructure that supports distributed migratable plastic user interfaces on clusters composed of a dynamic set of heterogeneous resources. It can be seen as an extension of the Aura approach where migration, distribution and adaptation are performed through a mix of open and close-adaptiveness [13], not at the whole application as proposed in Aura, but at any level of granularity. The functional components are supposed to run on top of I-AM, an interaction Abstract Machine under development [10].

I-AM can be viewed as a generalization of the X-window paradigm to support:

- The geometric relationships (topology) between physical surfaces,
- The projection of digital content onto the physical topology,
- The detection of arrival/departure of surfaces/instruments as well as of any change in the physical topology,
- The coupling of instruments with surfaces and information content,
- The elimination of hardware and O/S discrepancies.

I-AM is structured into three levels of abstraction sitting on top of the hardware and Operating Systems that form the legacy basis of a cluster: The Platform layer, the Logical Level layer, and the Interactor layer.

The *platform layer* provides the next level of I-AM, i.e., the Logical Level, with: (a) a normalized view of the set of physical interaction resources that are available on the elementary platforms of the cluster, and (b) networked communication means with those resources. More specifically, every processor that belongs to the cluster runs an IAMPlatformManager. An IAMPlatformManager is "elementary platform" centric: it manages the resources that are local to the platform it runs on. In order to support scalability and reconfigurability, it has no knowledge of the existence of the other platform managers of the cluster. Through contextors [6, 7], it discovers the interaction resources that are locally connected to the processor it runs on. It maintains a description of these resources (e.g., size, colour, shape, etc.), and it provides the world[2] with the basic means for using the interaction resources[3]. This includes (a) the publication of the existence of the interaction resources that may interest future consumers (e.g., I-AM applications), (b) communication ports that allow remote consumers to send requests to use a particular interaction resource, and if successful, to obtain dedicated communication ports for using the requested resource.

The *Logical Level* provides applications with a customized view of the physical platform layer. Applications that use an I-AM cluster may each have its own way to exploit and interpret the physical configuration of the cluster. Therefore, the Logical Level of I-AM allows applications to build their own view on top of the physical platform level. To get and exploit such a view, an application must create an instance of IAMApp. An IAMApp discovers the interaction resources[4] that are currently available in the cluster. To do so, an IAMApp expresses its interests through its ContextAdaptor. Expressions of interest include "give me a large surface", "give me the list of surfaces available in the cluster", "tell me when a new surface arrives or leaves" etc. A ContextAdaptor serves as a gateway between an application and the contextors that run in the environment to provide contextual information. Contextual information about the physical interaction resources is compiled from the information produced by the platform managers of the cluster. An IAMApp builds the communication channels with the interaction resources (surfaces) it is interested in. It maintains the geometric relationships between the physical surfaces that it is currently exploiting. It maps the physical topology onto the digital information content. To satisfy the customized view requirement, the Logical Level layer implements the mapping mechanism but uses the politics provided by the application.

The *Interactor level* implements the basic graphic concepts such as windows and widgets that populate the logical spaces managed by the Logical Level. An IAMInteractor provides the programmer with the conventional programming paradigm. As a result, an IAMInteractor can be created, destroyed, moved, etc. in a logical space. It has a position in the LogicalSpace, it has a height and width expressed in terms of logical pixels, etc. It hides away the facts that the interactor can migrate between physical surfaces, and that its rendering may overlap multiple surfaces. To do so, an IAMInteractor is (a) self-descriptive: it can save its state and serialize itself as an XML description. This description can be sent to a distinct IAMPlatformManager which can then reconstruct it appropriately on the target surface it manages, and (b) it is mapped into an EffectiveInteractor or, if its logical rendering overlaps N surfaces, it is mapped into N effective interactors. An EffectiveInteractor is an interactor that is effectively rendered on a physical surface. It makes concrete an IAMInteractor I. It sends to I the events it receives from the instruments attached to I. Conversely, any change in I is notified to its associated EffectiveInteractors.

I-AM is implemented in Java and supports a mix of PC's and Mac's. In its current implementation, I-AM is limited in the following ways: surfaces are screen displays, including video-projected displays. They are rectangular and assembled (coupled) within a plan. From the application programmer's perspective, windows, widgets and mouse pointers can migrate between screen displays as if they were connected to a single machine. In summary, I-AM provides the programmer with a uniform space of interaction resources managed by a cluster of heterogeneous elementary platforms. It extends the functional coverage of current windowing systems to distributed user interfaces across a dynamic set of interaction resources managed by a cluster of heterogeneous computers.

CONCLUSION

In this paper, we have presented new requirements that ubiquitous computing imposes on the software development of user interfaces so that they can be distributed dynamically across the resources of evolving computational clusters. In particular, we have stressed the necessity for abstracting away the heterogeneity of these resources into a normalized representation, and we have demonstrated the need for an explicit model of the geometrical relationships between these resources along with their physical properties. We have not discussed the consequences that these functional requirements put on computational perception. In particular, machine perception is needed to detect the arrival and departure of interaction resources, for eliciting their intrinsic properties as well as their topology. In addition, software engineering requirements on the very nature of the software components need to be made explicit. From our early experience, we can cite reflexivity, introspection

[2] « The world » denotes any software component that is not part of the IAMPlatformManager. I-AM applications are examples of such software components.

[3] Current limitation: only the existence of surfaces is published

[4] Current limitations: "interaction resources" should be understood as "surfaces" only.

and reconfigurability as well as languages for system architecture description [8].

ACKNOWLEDGMENTS

This work has been partly supported by the IST-FET GLOSS project (IST-2000-26070) and IST CAMELEON project (IST-2000-28323).

REFERENCES

1. Bederson, B., Meyer, J. Good, L. Jazz: An Extensible Zoomable User Interface Graphics Toolkit in Java. In Proceedings of UIST 2000. May 2000. p171-180.
2. Bier E., Freeman, S., Pier, K., The Multi-Device Multi-User Multi-Editor. In Proc. of the ACM conf. On Human Factors in Computer Human Interaction (CHI92), (1992), pp. 645-646.
3. Brumitt, B., Shafer, S. Better Living Through Geometry. Personal and Ubiquitous Computing 2001. Vol 5.1 Springer.
4. Calvary, G., Coutaz, J. Thevenin. D. A Unifying Reference Framework for the Development of Plastic User Interfaces. IFIP WG2.7 (13.2) Working Conference, EHCI01,Toronto, May 2001, Springer Verlag Publ., LNCS 2254, M. Reed Little, L. Nigay Eds, pp.173-192.
5. Coutaz, J., Lachenal, C., Dupuy-Chessa, S. Ontology for Multi-Surface Interaction, Proc. Interact 2003, Sept. 3-5, Zurich, IOS Press, 2003.
6. Coutaz, J. Rey, G. Foundations for a theory of Contextors. Proc. Of Computer-Aided Design of User Interfaces III, J. Vanderdonckt, C. Kolski Eds., Kluver Academic Publ., 2002, pp. 13-32.
7. Crowley, J., Coutaz, J., Rey, G., Reignier, P. Perceptual Components for Context-Aware Computing, UbiComp 2002:Ubiquitous Computing, 4th International Conference, Göteburg, Sweden, Sept./Oct. 2002, G. Borriello, L.E. Holmquist Eds., LNCS, Springer Publ., pp. 117-134.
8. Garlan, D., Monroe, R., Wile, D. Acme : Architectural Description of Component-Based Systems. Foundations of Component-Based systems, Gary T. Eds, Cambridge University Press, 2000, pp. 47-68
9. Hourcade, J., Bederson, B. Architecture and Implementation of a Java Package for Multiple Input Devices (MID). 1999. Disponible à l'adresse : http://www.cs.umd.edu/hcil/mid/
10. Lachenal, C., Rey, G., Barralon, N. MUSICAE, an infrastructure for MUlti-Surface Interaction in Context Aware Environment. In Proc. HCI International, Crète, June 2003,pp. 125-126
11. Luyten, K., Vandervelpen, K. Coninx, K. Migratable user interfaces Descriptions in Component-Based Development. DSV-IS 2002, Rostock, Springer Verlag Publ., 2002
12. Myers, B., Stiel, H., Gargiulo, R. Collaboration Using Multiple PDAs Connected to a PC. In Proceedings CSCW'98: ACM Conference on Computer-Supported Cooperative Work, 1998, Seattle, WA. pp. 285-294.
13. Oreizy, P., R. Tay lor, R. et al. An Architecture-Based Approach to Self-Adaptive Software. In IEEE Intelligent Systems. pp. 54-62, May-June, 1999.
14. Paternò, F., Santoro, C. One model, many interfaces, in Proc. Computer-Aided Design of User Interfaces III (CADUI), J. Vanderdonckt, C. Kolski Eds., Kluver Academic Publ., 2002.
15. Rauterberg, M. et al. BUILT-IT: A Planning Tool for Consruction and Design. In Proc. Of the ACM Conf. In Human Factors in Computing Systems (CHI98) Conference Companion, (1998), pp. 177-178
16. Rekimoto, J. Pick-and-Drop: A Direct Manipulation Technique for Multiple Computer Environments. In Proceedings of UIST'97, ACM Publ., 1997, pp. 31-39.
17. Rekimoto, J., Masanori, S. Augmented Surfaces: A Spatially Continous Workspace for Hybrid Computing Environments. Proceedings of CHI'99, ACM publ., 1999.
18. Sousa, J., Garlan, D. Aura : an Architectural Framework for User Mobility in Ubiquitous Computing Environments. IEEE-IFIP Conf. on Software Architecture, Montreal, 2002
19. Streitz, et al. I-LAND: An interactive Landscape for Creativity and Innovation. In Proceedings of CHI'99, ACM publ.
20. Tandler, P. Software Infrastructure for Ubiquitous computing Environments : Supporting synchronous Collaboration with Heterogenous devices. In Proceedings of UbiComp 2001, Springer Publ..
21. Thevenin, D., Coutaz, J. Plasticity of User Interfaces: Framework and Research Agenda. In Proc. Interact99, Edinburgh, , A. Sasse & C. Johnson Eds, IFIP IOS Press Publ. , 1999, pp.110-117.
22. Want, et al. The Personnal Server : The Center of Your Ubiquitous World. Intel Research White Paper May 2001.

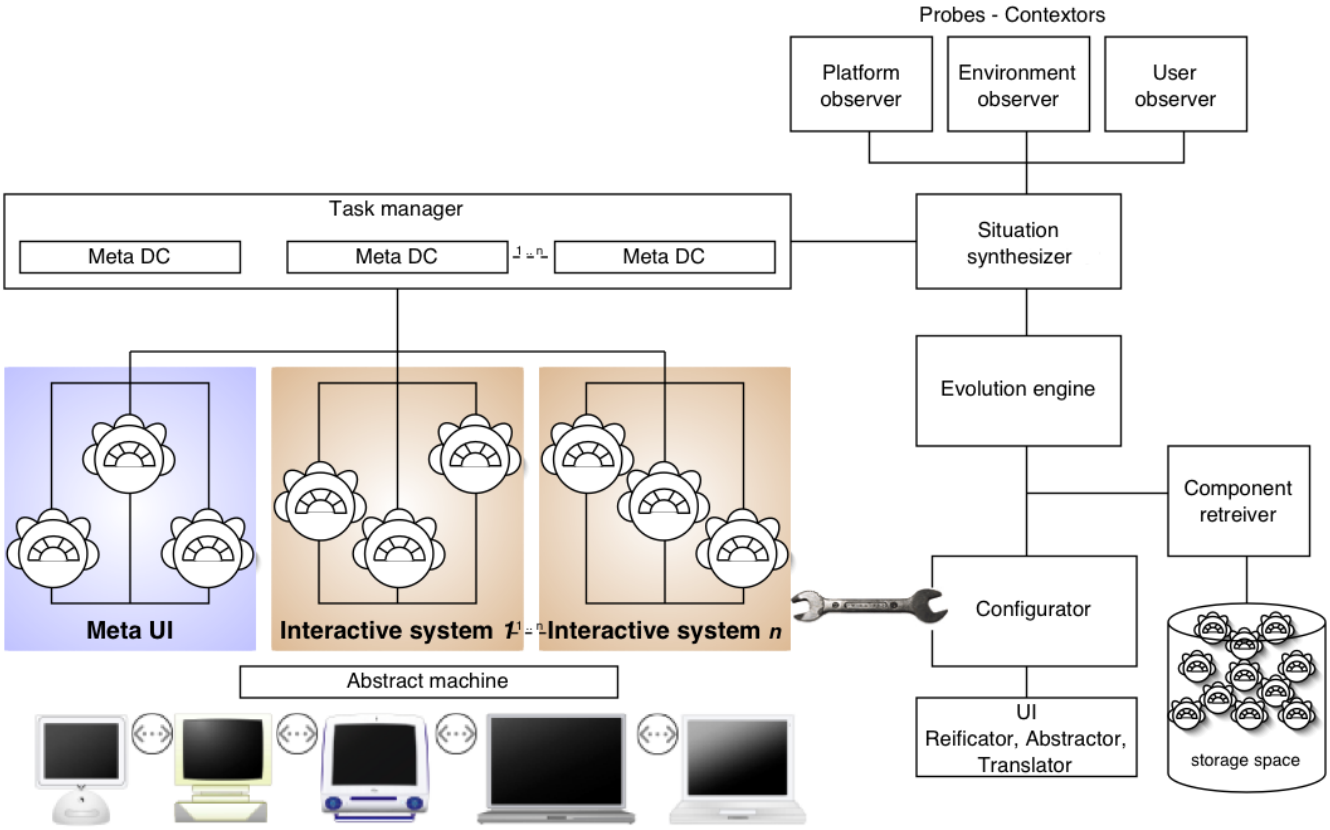

Figure 1. The CAMELEON overall middleware infrastructure that supports distributed, migratable and plastic UI's.