

CAMELEON-RT: a Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces

Lionel Balme, Alexandre Demeure, Nicolas Barralon, Joëlle Coutaz, Gaëlle Calvary

CLIPS-IMAG, Université Joseph Fourier
BP 53, 38041 Grenoble Cedex 9, France
{lionel.balme, alexandre.demeure, nicolas.barralon,
joelle.coutaz, gaelle.calvary}@imag.fr
<http://www-clips.imag.fr/iim/>

Abstract. This paper defines the problem space of distributed, migratable and plastic user interfaces, and presents CAMELEON-RT¹, a technical answer to the problem. CAMELEON-RT¹ is an architecture reference model that can be used for comparing and reasoning about existing tools as well as for developing future run time infrastructures for distributed, migratable, and plastic user interfaces. We have developed an early implementation of a run time infrastructure based on the precepts of CAMELEON-RT¹.

1 Introduction

Technological advances in computing, sensing, and networking, are rapidly leading to the capacity for individuals to create and mould their own interactive spaces. Interactive spaces will be assembled opportunistically from public hot spots and private devices to provide access to services within the global computing fabric. Interactive spaces will also take the form of autonomous computing islands whose horizon will evolve, split and merge under the control of users. With this move to ubiquitous computing, user interfaces (UI) are no longer confined to a unique desktop. Instead, UI's may be distributed and migrate across a dynamic set of interaction resources that are opportunistically composed, borrowed and lent. As a consequence of distribution and migration, user interfaces must be plastic in order to adapt gracefully to changes of the interactive space.

To address the problem of developing distributed, migratable, plastic UI's, we propose CAMELEON-RT, a conceptual architecture reference model. This model is a canonical functional decomposition that can be used for comparing and reasoning about existing tools as well as for developing future run time infrastructures for distributed, migratable, and plastic user interfaces (DMP-UI). The article is structured as follows. In the next section, we introduce the terminology to establish a common understanding of the problem space for DMP-UI. We then analyze the state of the art

¹ RT stands for Run-Time.

in the light of the problem space to motivate our own proposition with CAMELEON-RT and two case studies. We close the presentation with a discussion for future work.

2 Terminology and Problem Space

Our terminology covers two aspects of the problem: the capacity for people to mould the digital and the physical worlds into cohesive *interactive spaces*, and the consequence on user interfaces which, from centralized and immutable, become *distributed*, *migratable*, and *plastic*.

2.1 Interactive space

An *interactive space* is a combination of three complementary things. It includes a) the physical place where the interaction takes place, b) the computing, networking and interaction resources available at this place, and c) the digital world (or set of services) that supports human activities in this space. The physical place is modeled with attributes and functions such as location, social use, and light conditions. The computing, networking and interaction resources bind together the physical space with the digital world. In particular, an *interaction resource* is a physical entity that allows users to modify and/or observe the state of the digital world. Typically, mice and keyboards are used as input interaction resources (we call them *instruments*), but phicons are instruments as well. Display screens are used as output interaction resources (we call them *surfaces*). Augmented tables and rain curtains are surfaces as well. The interaction resources of an interactive space are managed by a platform.

The *platform* is *elementary* when it is composed of one computer. It is a *cluster* when it is assembled from a set of computers. The assembly may be *static* (the configuration cannot be modified on the fly) or *dynamic*. When dynamic, an elementary platform or an interaction resource may arrive or disappear. Alternatively, the set of interaction resources may stay the same but the relationships between them, such as the orientation of the surfaces, may change. The cluster is *homogeneous* (it is composed of identical elementary platforms) or *heterogeneous* (the resources and/or the operating system of the constituents differ).

For example, the I-land's DynaWall [19] is a static homogeneous cluster: it is composed of three interconnected electronic whiteboards controlled with the same underlying infrastructure, Beach[20]. On the other hand, the ConnectTables of I-land, where two identical tablets running the same system can be plugged together, constitute a dynamic homogeneous cluster [21]. Pebbles [12] supports the construction of dynamic heterogeneous clusters where multiple PDA's can be connected on the fly to control the display surface of a workstation. In iRoom [9], the cluster is a dynamic assembly of workstations that runs a mix of Windows and Unix. Within an interactive space, we need to consider how UI's are distributed, how they can migrate and support plasticity.

2.2 UI Distribution

A *UI is distributed* when it uses interaction resources that are distributed across a cluster. For example, in graphical UI's (GUI), the rendering is distributed if it uses surfaces that are managed by different elementary platforms. The granularity of the distribution may vary from application level to pixel level.

At the *application level*, the GUI is fully replicated on the surfaces managed by each elementary platform. The x2vnc implementation of the VNC protocol offers an application level distribution. At the *workspace level*, the unit for distribution is the workspace. A workspace is a compound interactor that supports the execution of a set of logically connected tasks. PebblesDraw [12] and Rekimoto's Pick and Drop [17] are examples of UI distribution at the workspace level. The *interactor level* distribution is a special case of the workspace level where the unit for distribution is an elementary interactor. At the *pixel level*, any user interface component can be partitioned across multiple surfaces. For example, in the DynaWall, a window may simultaneously lie over two contiguous white boards as if these were managed by a single computer.

2.3 UI Migration

UI migration corresponds to the transfer of all or part of the UI to different interaction resources whether these resources belong to the current platform or to another one.

Migration is *static* when it is performed off-line between sessions. It is *dynamic* when it occurs on the fly. In this case, a state recovery mechanism is needed so that users can pursue their activities in a seamless manner. In addition, the migration of a user interface is *total* if the user interface moves entirely to a different platform. It is *partial* when a subset only of the user interface moves to different interaction resources. For example, on the arrival of a PDA, the control panels currently rendered on a whiteboard migrate to the PDA.

Migration and distribution are two independent notions: a UI may be distributed but not migratable. A centralized UI may migrate to a cluster and distributes itself across the interaction resources of the new platform. A priori, the most powerful UI's are those that are both dynamically distributable and migratable. However, migration and distribution may in turn require the UI to be plastic.

2.4 UI Plasticity

The term *plasticity* is inspired from the capacity of solids and biological entities such as plants and brain, to adapt to external constraints to preserve continuous usage. Applied to HCI, *plasticity* is the capacity of an interactive system to adapt to *changes of the interactive space* while *preserving usability* [2]. Usability is defined as a set of properties $\{p_1, \dots, p_i, \dots, p_n\}$ (e.g., observability, predictability [7]) such that, for each p_i , a metrics is defined with a domain of values d_i). The job of a plastic UI is to maintain the set of properties within their domain of values. Given a user interface *UI* and

its usability $U = \{(p_1, d_1), \dots, (p_i, d_i), \dots, (p_n, d_n)\}$, the *domain of plasticity* of *UI* is the set of situations S for which *UI* is able to preserve U .

Early work on UI plasticity demonstrates that UI migration and distribution between very different platforms go far beyond software portability. For example, the static migration on a mobile phone of an application designed for a workstation may result in multiple forms of adaptation. This may range from replacing graphics interactors with vocal interactors, to restructuring the dialogue or even suppressing services [22]. Retargeting a UI may be static and/or performed at run time. When retargeting is static, a set of *pre-computed concrete user interfaces* (CUI) is produced in advance for a predefined set of situations.

The production of CUI's by *reification* is one approach to the problem: typically, the process starts from high-level descriptions such as task and domain models to produce an abstract UI (AUI), then from an AUI, produces one or multiple CUI's. Alternatively, an existing CUI may be reverse-engineered by means of *abstraction* to obtain an AUI and/or a task model. These abstract representations are then *translated* to fit the new target, then reified into new CUI's.

Having defined the dimensions of the problem space for DMP-UI's, we need to identify how the software tools of the state of the art address the problem.

3 Analysis of the State of the Art

UI plasticity is supported for centralized, statically migratable UI's only. No tool addresses UI distribution and no tool supports on-the-fly migration of the user interface between platforms. Migration can only occur between sessions. Development tools like Teresa [15] and ArtStudio [2] pre-compute CUI's from high level specifications. They are completed with reverse-engineering tools like Webrevence [14] and Vaquita [23] that proceed with a combination of abstraction, translation, and reification. Digymes [4] and Icraft [16], on the other hand, generate CUI's at run time where a renderer dynamically computes a CUI from an AUI.

Websplitter[8] supports the distribution of web pages content at the interactor level across the interaction resources of heterogeneous clusters, but distribution is statically specified in an XML policy file. In Pebbles, the types of interaction resources are known and the distribution of the UI is statically assigned based on this knowledge: the public screen of a workstation contains a shared workspace whereas PDA's contain a panel to control the shared display. In iRoom, mouse pointers can migrate between the screens of the cluster. However windows are confined to the screen where they have been created. Beach, on the other hand, supports the dynamic distribution and migration of UI's at the pixel level, but the cluster is static and homogeneous.

This brief overview of the state of the art reveals that no software tool currently supports all aspects of distributed, migratable and plastic user interfaces. With CAMELEON-RT, we propose an architecture reference model that integrates all of the functional components necessary to support the dynamic distribution, migration and plasticity of UI's across dynamic heterogeneous clusters. This canonical conceptual architecture can then be instantiated in different ways to implement run-time

infrastructures adequate for a subset of the problem space of DMP-UI's. This is what we have done with the development of two very different case studies.

4 Case Studies: CamNote and I-AM

CamNote (for CAMELEON Note) is a slides viewer that runs on a dynamic heterogeneous platform. This platform may range from a single PC to a cluster composed of a PC and a PDA. I-AM is a platform manager similar in spirit to X-Window, but that supports dynamic heterogeneous clusters of workstations.

4.1 CamNote

The UI of CamNote includes four workspaces: a slides viewer, a note editor for associating comments to slides, a video viewer also known as pixels mirror that shows a live video of the speaker [24], and a control panel to browse the slides and to setup the level of transparency of the pixels mirror. As shown in Figure 1, the pixels mirror is combined with the slides viewer using alpha-blending. Speakers can point at items on the slide using their finger. This means of pointing is far more compelling and engaging than the conventional mouse pointer that no one can see [11].

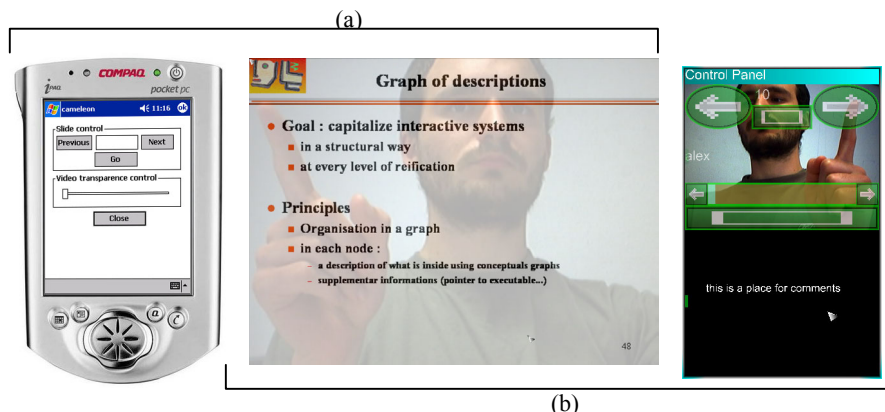


Fig. 1. (a) The user interface of CamNote when distributed on a PC and a PocketPC screens; (b) the control panel when displayed on the PC screen

Figure 1a shows a Pebbles-like configuration where the graphical UI is distributed across the surfaces of a PC and of a PDA. The slides viewer is displayed in a rotative canvas so that it can be oriented appropriately when projected on an horizontal surface (e.g., a table). If the PDA disappears from the cluster, the control panel automatically migrates to the PC screen. Because different resources are now available, the panel is plastified. As shown in Figure 1b, the retargeted control panel includes different interactors, but also a miniature representation of the speaker's video is now

available. During the migration-retargeting process, users can see the control panel emerging progressively from the slides viewer while rotating so that they can evaluate the state of the transition. The UI, which was distributed on an heterogeneous cluster is now centralized on an elementary platform. The new UI results from a dynamic partial migration and retargeting at the workspace level. Conversely, if the PDA re-enters the platform, the UI automatically switches to the configuration of Figure 1a) and the control panel disappears from the PC screen by weaving itself into the slides viewer before reappearing on the PDA.

4.2 The Interaction Abstract Machine (I-AM)

I-AM (Interaction Abstract Machine) supports the dynamic configuration of interaction resources to form a single logical interactive space [1]. These resources are managed by different elementary workstations running distinct operating systems (i.e., MacOS X, Windows NT and XP). Users can distribute and migrate user interfaces at the pixel level as if these UI's were handled by a single computer. This illusion of a unified space is provided at no extra cost for the developer who can re-use the conventional GUI programming paradigm.

Figure 2 shows early examples of interaction techniques that allow users to control the platform of their interactive space. Figure 2a corresponds to the situation where two applications are running on two independent workstations. A closed blue border outlines the screens to denote the absence of coupling. In Figure 2b, the screens are now coupled to provide the "single display area" function. A blue border outlines the display area and a gateway shows where interactors can transit between the screens.

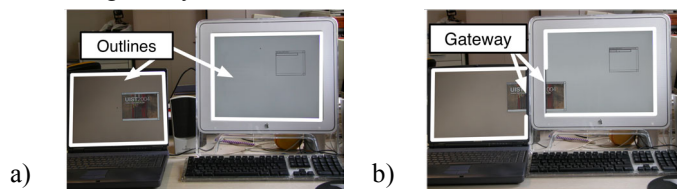


Fig. 2. (a) The PC and the Macintosh are decoupled and run two applications. (b) The two screens are coupled by bringing them in close contact to form a single information space. (Outlines have been artificially enhanced on the pictures to increase readability.)

Within an interactive space, any instrument can be used to modify any interactor. For example, in the configuration of Figure 2b, a PC mouse can be used to move a window created on the Macintosh and migrate it to the PC. Or the two mice can be used simultaneously. The user can select a text field interactor displayed on the Macintosh screen with the PC mouse. Text can now be entered with the PC keyboard. If the text field is selected with the Macintosh mouse, text can be entered with the Macintosh keyboard as well. I-AM supports the dynamic configuration of clusters of workstations running different operating systems, it supports the dynamic migration and distribution of UI at the pixel level at no extra cost for the programmer, but it

does not support plasticity. I-AM and CamNote, although very different in terms of their functional coverage, comply with the principles of CAMELEON-RT.

5 The CAMELEON-RT Architecture Reference Model

As shown in Figure 3, CAMELEON-RT is structured into three levels of abstraction: at the two extremes, the interactive systems layer and the platform layer; at the core of the architecture, the Distribution-Migration-Plasticity middleware (DMP-middleware) that provides mechanisms and services for DMP UI's.

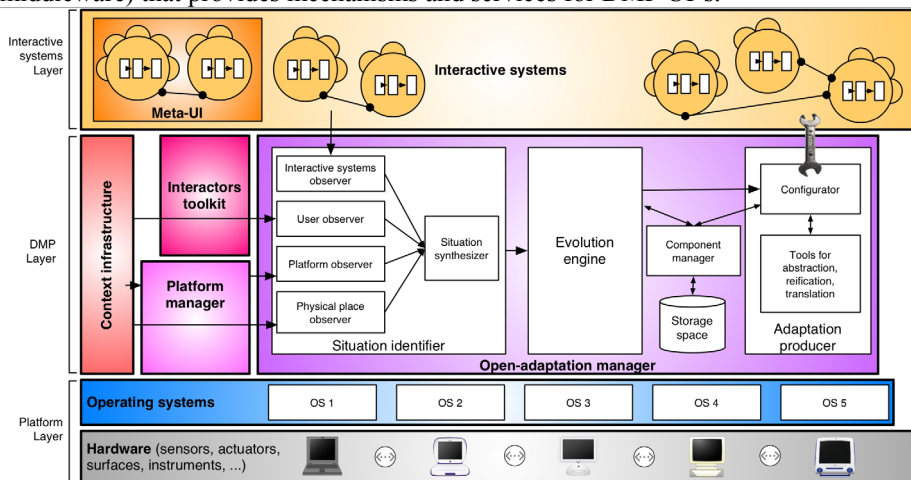




Fig. 3. The CAMELEON-RT architecture reference model. A flower-like shape, , denotes open-adaptive components. The miniature adaptation-manager shape, , denotes close-adaptive components. Arrows denote information flow, and lines bi-directional links.

5.1 The Platform Layer

The platform layer corresponds to the notion of platform as defined in Section 2. It includes the hardware and the legacy operating system(s), which, together, form the ground-basis of an interactive space. The hardware denotes a wide variety of physical entities: surfaces and instruments, computing and communication facilities, as well as sensors and actuators.

5.2 The Interactive Systems Layer

This layer includes the interactive systems (e.g., CamNote) that users are currently running in the interactive space. The Meta-User Interface (meta-UI) is one of them.

The *meta-user interface* is to interactive spaces what the desktop is to conventional workstations: it binds together the activities that can be performed within the interactive space and provides users with the means to configure, control and evaluate the state of the space. In practice, the meta-UI brings together the user interfaces of all of the DMP-middleware components in a unified manner. For example, the interaction techniques used to couple two surfaces is part of the meta-UI. In CamNote, the animation that allows users to evaluate the migration and adaptation process of the control panel is also part of the meta-UI. As for the UI of any application running in the interactive space, a meta-UI is DMP.

As discussed in 2.4, a DMP UI is characterized by a domain of plasticity. This means that the UI embeds mechanisms for self-adaptation as long as the requirements for adaptation lie within its domain of plasticity. The UI is said to be *close-adaptive* for these situations. For situations that cannot be handled by the UI alone, the UI must be *open-adaptive* so that the DMP-middleware layer can take the process over. The UI is open-adaptive if it provides the world with management mechanisms. Management mechanisms include self-descriptive meta-data (such as the current state and the services it supports and requires), and the methods to control its behavior such as start/stop and get/set-state. Software reflexivity coupled with a component model is a good approach to achieve open-adaptiveness [13]. Close-adaptiveness and open-adaptiveness both comply with the four-step process presented in 5.3.4: observe the world, detect situations that require adaptation, compute a reaction that satisfies the situation, and generate a new UI.

5.3 The DMP-middleware layer

The DMP-middleware layer aims at satisfying three classes of requirements of our problem space: modeling the physical space, supporting dynamic heterogeneous clusters, and UI adaptation when distribution and migration occur. To each of these requirements corresponds a service of the DMP-layer: a context infrastructure, a platform manager along with its interaction toolkit, and an open-adaptation manager.

5.3.1 The Context infrastructure

The context infrastructure allows the interactive space to build and maintain a model of the physical place. From sensor data as well as from low-level operating system events, the context infrastructure generates contextual information at the appropriate level of abstraction. The Context Toolkit [6] and the Contextors [3] are example of tools that can be used to implement a context infrastructure.

5.3.2 The Platform Manager and Interaction Toolkit

The platform manager and the interaction toolkit play the same functional role as the X-window environment, but extended to dynamic heterogeneous clusters. This includes: a) supporting resource discovery, b) hiding the heterogeneity of operating systems and hardware, and possibly, c) supporting the distribution and migration of UI's. The interaction toolkit may be a conventional toolkit such as Motif or Swing,

and/or post-WIMP toolkits such as DiamondSpin [18] and our own toolkit used in CamNote to develop rotative and zoomable UI's.

Pebbles and iRoom support requirements a) and b), whereas c) is not addressed or is limited to pointing instruments. In Aura, a context infrastructure is used for resource discovery (as well as for modeling the physical place). On the other hand, Aura which addresses elementary platforms only, re-uses the native windowing systems and toolkits as platform managers and toolkits. Beach supports b) and c) for homogeneous clusters only. However, it is unclear whether Beach is able to detect the arrival/departure of elementary platforms.

I-AM covers the a), b) and c) requirements. It uses the Contextors infrastructure to discover changes of the platform. To support the migration of UI's at the pixel level, I-AM maintains one logical space per interactive system. A logical space is an abstract drawable populated with logical interactors (i.e., those that the programmer has created with the interaction toolkit). Logical interactors are projected on the surfaces of the cluster into physical interactors. The projection is an affine transformation that takes into account the geometrical relationships between the surfaces as well their resolution. For example, a logical interactor that lies over two surfaces is projected into two physical interactors, one per surface. For input, I-AM redirects input events performed on physical widgets, to the logical interactors that own them.

5.3.4 The Open-Adaptation Manager

The Open-Adaptation Manager is a key component of CAMELEON-RT. It includes observers that feed into a situation synthesizer with appropriate contextual information. The situation synthesizer informs the evolution engine of the occurrence of a new situation that may require an open adaptation. If so, the evolution engine uses the components retriever and a configurator to produce a new UI.

Observers serve as gateways between the "world" and the situation synthesizer. The *platform observer* gathers information about the platform (e.g., a new PDA has arrived/has left, two surfaces have been coupled) by subscribing to the components of the context infrastructure that probes the evolution of the platform. The *physical place observer* maintains a model of the physical place (e.g., we are in room R, or in street S), and the *users observer* probes users (for instance their profile, or their position relative to a wall surface such that information is not projected in their back). The *interactive systems observer* subscribes to information relevant to interactive systems plastification. For instance, an interactive system may produce the event "current situation S is out of my domain of plasticity" so that opne-adaptation can take over the retargeting process.

The *situation synthesizer* computes the current situation from information (i.e., the observables) provided by the observers. A situation is defined by a set of observables that satisfies a set of predicates [5]. When the cardinality of the set of observables changes and/or when the predicates do not hold anymore, we enter a new situation. For example, in CamNote, the arrival or departure of a PDA results in a new situation. Situations form a graph. Ideally, the graph of situations results from a mix of specifications provided by developers, by users (using the meta-UI), or learnt automatically by the situation synthesizer. In CamNote, the graph of situations has been

provided by the programmer. Entering a new situation is notified to the evolution engine.

The *evolution engine* elaborates a reaction in response to the new situation. As for the graph of situations, the reaction may be a mix of specifications provided by developers and/or users (using the meta-UI), or learnt by the evolution engine. In CamNote, reactions are expressed by developers in terms of rules. For example, “if a new PDA arrives, move the control panel to the PDA”. The reaction may result in retargeting all or part of the UI. If so, the evolution engine identifies the components of the UI that must be replaced and/or suppressed and provides the configurator with a plan of actions. In the case of CamNote, the plan is to “replace the PC control panel with a PDA control panel without losing any previous work”.

The *Configurator* executes the plan. If new components are needed, these are retrieved from the *components storage* by the *components Manager*. In CamNote, we reuse a technique developed in Information Retrieval: components of the components storage are described with conceptual graphs and retrieved with requests expressed with conceptual graphs. By exploiting component reflexivity, the configurator stops the execution of the “defectuous” components specified in the plan, gets their state, then suppresses or replaces them with the retrieved components and launches these components based on the saved state of the previous components. In CamNote, the PC control panel is replaced with a PDA control panel and its state is restored properly so that users can continue the slides show at the exact slide number before migration occurred.

The components referred to in the action plan do not necessarily exist as executable code. They may instead be high-level descriptions such as task models or AUI’s. If so, the configurator relies on *reifiers* to produce executable code as in Digymes and iCrafter. A retrieved component may be executable, but may not fit the requirements. It may thus be reversed-engineered through *abstractors*, and then transformed by *translators* and reified again into executable code [23].

5.4 Discussion

CAMELEON-RT is a functional decomposition that covers all aspects of DMP UI’s. It is not an implementational architecture. In particular, we do not address the allocation of functions across processes and processors, and we leave open the choice of architecture styles. However, the nature of ubiquitous computing (e.g., platform heterogeneity and dynamicity, interactive islands based on ad-hoc networks), suggests the following heuristics: apply the principles of exo-kernels by making a clear distinction between core functions and extension functions that can be called upon dynamically and possibly remotely. By doing so, low-end elementary platforms can be addressed. Replace the client-server model with a P2P architecture style so that ad-hoc interactive islands can be constructed on the fly. Use a reflexive component-connector approach to support software reconfiguration. Our early experience with the implementation of the Contextors infrastructure, I-AM, and CamNote, demonstrate that these rules are viable.

6 Conclusion

The analysis of the state of the art shows that current research projects address a small subset of the problem space of distributed, migratable, and plastic user interfaces. The CAMELEON-RT model provides software designers with a general framework that addresses both small and large scales computing environments, as well as all forms of UI distribution, migration and plastification. In addition, it makes explicit the notion of meta-user interface, an emerging notion found implicitly in the literature in expressions like “how users will configure and control their environment?”. We propose early heuristics to facilitate the exploitation of CAMELEON-RT for the practical deployment of run time infrastructures. These need to be refined and CAMELEON-RT must be evaluated with further experiments.

Acknowledgments

This work has been partly supported by the FET GLOSS project (IST-2000-26070), CAMELEON (IST-2000-28323) and FAME (IST-2002-28323).

References

1. Coutaz, J., Lachenal, C., Dupuy-Chessa, S. Ontology for Multi-surface Interaction. Proc. Interact 2003, M. Rauterberg et al. Eds, IOS Press Publ., IFIP, 2003, pp.447-454.
2. Calvary G., Coutaz J., Thevenin D., A Unifying Reference Framework for the Development of Plastic User Interfaces, EHCI01, Toronto, Springer Verlag Publ., LNCS 2254, M. Reed Little, L. Nigay Eds, May 2001. pp.173-192.
3. Coutaz J., Rey G., Foundations for a theory of Contextors, Proc. Of Computer-Aided Design of User Interfaces III, J. Vanderdonck, C. Kolski Eds., Kluwer Academic Publ., 2002, pp. 283-303.
4. Coninx K., Luyten K., Vandervelpen C., Van den Bergh J., Creemers B., Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems.
5. Crowley, J., Coutaz, J., Rey, G., Reignier, P. Perceptual Components for Context-Aware Computing, UbiComp 2002: Ubiquitous Computing, 4th International Conference, Göteborg, Sweden, Sept./Oct. 2002, G. Borriello, L.E. Holmquist Eds., LNCS, Springer Publ., 2002, pp. 117-134.
6. Dey, A. K. “Understanding and using context”, Personal and Ubiquitous Computing, Vol 5, No. 1, pp 4-7, 2001.
7. Gram, C., Cockton, G., Design Principles for Interactive Software, produce by IFIP WG 2.7 (13.4), C. Gram & G. Cockton Eds., Chapman&Hall Publ., 1996
8. Han R., Perret V., Naghshineh M., WebSplitter: A Unified XML Framework for Multi-Device Collaborative Web Browsing, Appeared in ACM Conference on Computer Supported Cooperative Work (CSCW) 2000.
9. Johanson B., Fox A., Winograd T., The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms, IEEE Pervasive Computing Magazine 1(2), April-June 2002, pp. 71-78.

10. Luyten, K., Vandervelpen, C., Coninx, K. Migratable user interfaces Descriptions in Component-Based Development. DSV-IS 2002, Rostock, Springer Verlag Publ., 2002.
11. Morikawa, O., Maesako, T. HyperMirror : Toward Pleasant-to-use Video Mediated Communication System. In proceedings of CSCW'98, ACM Publ., Seattle, Washington USA. pp. 149-158
12. Myer B. A., Using Handhelds and PCs Together, Communication of the ACM, Vol. 44, No 11, November 2001, pp. 34-41.
13. Oreizy, P., Taylor, R., et al. An Architecture-Based Approach to Self-Adaptive Software. In IEEE Intelligent Systems, May-June, 1999, pp. 54-62.
14. Paganelli L., Paternò F., Automatic Reconstruction of the Underlying Interaction Design of Web Applications, Proceedings of SEKE 2002 (Ischia, Italy, July 2002).
15. Paternò, F., Santoro, C. One model, many interfaces. In Proc. CADUI 2002, J. Vanderdonckt, C. Kolski Eds., Kluwer Academic Publ., 2002.
16. Ponnekanti, S., Lee, B., Fox, A., Hanrahan, P., Winograd, T. Icraft: a Service Framework for Ubiquitous Computing Environments. In Proc. Ubicomp 2001, G. Abowd, B. Brumitt, S. Shafer Eds., Springer Publ., LNCS 2201, 2001, pp. 57-75.
17. Rekimoto, J. Pick-and-Drop: A Direct Manipulation Technique for Multiple Computer Environments. In Proceedings of User Interface Software and Technology (UIST 1997), ACM, 1997, pp. 31-39.
18. Shen C., Vernier F., Forlines C., Ringel M., DiamondSpin: An Extensible Toolkit for Around-the-Table Interaction. In proc. CHI 2004, April 24–29, 2004, Vienna, Austria.
19. Streitz, N., Geibler, J., Holmer, T., Konomi, S., Müller-Tomfelde, C., Reischl, W., Rexroth, P., Seitz, P., Steinmetz, R. i-LAND: An interactive Landscape for Creativity and Innovation. In Proc. of the ACM conf. On Human Factors in Computer Human Interaction (CHI99), ACM, 1999, pp. 120-127.
20. Tandler, P. Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogenous Devices. In Proc. Ubicomp 2001, Atlanta Sept. 2001, Abowd G., Brumitt B. and Shafer S. Eds., Springer, LNCS 2201, pp. 96-115.
21. Tandler, P., Prante, T., Müller-Tomfelde, C., Streitz, N., Steinmetz, R. ConnecTables: Dynamic Coupling of Displays for the Flexible Creation of Shared Workspaces. In Proc. UIST 2001, ACM publ., 2001, pp. 11-20.
22. 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.
23. Vanderdonckt, J., Bouillon, L., and Souchon, N., Flexible Reverse Engineering of Web Pages with Vaquita. In Proc. WCRE'200: IEEE 8th Working Conference on Reverse Engineering. Stuttgart, October 2001. IEEE Press.
24. Vernier, F., Lachenal, C., Nigay, L., Coutaz, J. Interface Augmentée Par Effet Miroir, in Proc. IHM'99. (AFIHM conference on Human-Machine Interface, 22-26 November 1999 Montpellier, France), Cepadues Publ., pp. 158-165.