An MDE-SOA Approach to Support Plastic User Interfaces in Ambient Spaces

J. Coutaz, L. Balme, X. Alvaro, G. Calvary, A. Demeure, J.-S. Sottet

Université Joseph Fourier, Grenoble
E-mail: {Joelle.Coutaz, Lionel.Balme, Xavier.Alvaro, Gaelle.Calvary, Alexandre.Demeure, Jean-Sebastien.Sottet}@imag.fr

Abstract. User interface (UI) plasticity denotes UI adaptation to the context of use (user, platform, physical and social environment) while preserving usability. Our approach to this problem is to bring together MDE (Model Driven Engineering) and SOA (Service Oriented Approach) within a unified framework that covers both the development stage and the runtime phase of plastic UI's. In particular, an interactive system is modelled as a graph of models that can be dynamically manipulated by, and/or encapsulated as services.

Keywords: User Interface plasticity, UI adaptation, Model-driven Engineering (MDE), Service Oriented Architecture (SOA), context of use, meta-UI.

1 Introduction

With the 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 result, user interfaces must adequately be plastic.

UI plasticity denotes the capacity for user interfaces to adapt to the context of use while preserving usability [7]. The *context of use* is a structured information space whose finality is to inform the adaptation process. It includes a model of the user who is intended to use (or is actually using) the system, the social and physical environments where the interaction is supposed to take place (or is actually taking place), and the platform to be used (or is being used). The latter covers the set of computing, sensing, communication, and interaction resources that bind together the physical environment with the digital world. *Usability* expresses the *useworthiness* of the system: the value that this system has in the real world [8].

From the software perspective, UI plasticity goes far beyond UI portability and UI translation. As discussed in Section 2, the problem space is complex. Software adaptation has been addressed using many approaches over the years, including Machine Learning [14], Model-Driven Engineering (MDE) [18], and Component-

oriented services [17]. Our approach to the problem of UI plasticity is based on the following observations: first, every paradigm has its own merits targeted at specific requirements. Thus, in an ever-changing world, one single approach is doomed to failure. Second, software tools and mechanisms tend to make a dichotomy between the development stage and the runtime phase making it difficult to articulate run-time adaptation based on semantically rich design-time descriptions.

Our approach to UI plasticity is to bring together MDE (Model Driven Engineering) and SOA (Service Oriented Approach) within a unified framework that covers both the development stage and the runtime phase of interactive systems. The principles of our solution space are presented in Section 3.

2 The Problem Space of UI Plasticity

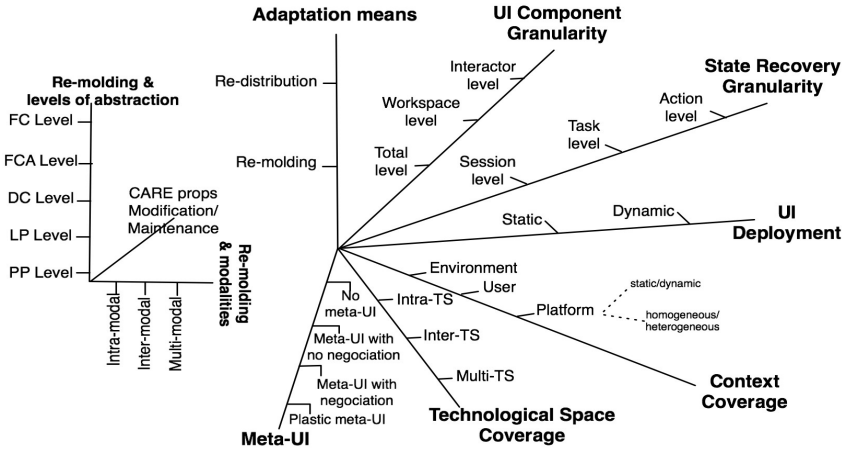


Fig. 1. The problem space of UI plasticity.

As shown in Fig. 1, the problem space of UI plasticity is characterized (but is not limited to) the following dimensions: the means used for adaptation (i.e. re-molding and re-distribution); the smallest UI units that can be adapted by the way of these means (from the whole UI considered as a single piece of code to the finest grain: the interactor); the granularity of state recovery after adaptation has occurred (from the session level to the user's last action); the UI deployment (static or dynamic) as a way to characterize how much adaptation has been pre-defined at design-time VS computed at runtime; the context coverage to denote the causes for adaptation with which the system is able to cope; the coverage of the technological spaces as a way to characterize the degree of technical heterogeneity supported by the system; and the existence of a meta-UI to allow users to control and evaluate the adaptation process. A subset of these dimensions is now developed in detail.

2.1 Two Adaptation Means: UI Re-molding and UI Re-distribution

UI re-molding denotes the reconfiguration of the user interface that is perceivable to the user and that results from the application of transformations on the user interface. These transformations may result in suppressing the UI components that become irrelevant in the new context of use; inserting new UI components to provide access to new services, or reorganizing the UI components by revisiting their spatial layout and/or their temporal dependency. Re-molding may result in using different modalities, or in exploiting multimodality differently. For example, because of the lack of computing power, the synergistic-complementarity [9] of the source multimodal UI (as in "put that there") may be transformed into an alternate-complementarity (the user has to behave seuquentially), or complementarity itself may disappear.

UI re-molding is *intra-modal* when the source UI components that need to be changed are retargeted within the same modality. It is *inter-modal* when the source UI components that need to be changed are retargeted into a different modality. Inter-modal retargeting may engender a modality loss or a modality gain. Thus, a source multimodal UI may be retargeted into a mono-modal UI and conversely, a mono-modal UI may be transformed into a multimodal UI. As in Teresa [2], re-molding is *multi-modal* when it uses a combination of intra- and inter-modal transformations.

UI adaptation is often assimilated to UI re-molding. This is true as long as we live in a closed world where the interaction resources are limited to that of a single computer at a time. In ubiquitous computing, the platform may be a dynamic cluster composed of multiple interconnected computing devices whose interaction resources, all together, form a *habitat* for UI components. In this kind of situation, instead of being *centralized*, the user interface may be *distributed* across the interaction resources of the cluster. *UI re-distribution* denotes the re-allocation of the UI components of the system to different interaction resources. For example in CamNote, on the arrival of a PDA, the control panel of the slides-show dynamically migrates from the PC to the PDA (See Annex 1). Note that, as a consequence of re-distribution, all or parts of the UI may require re-molding.

2.2 Coverage of the Technological Spaces

"A technological space is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities." [12]. Examples of technological spaces include documentware concerned with digital documents expressed in XML, dataware related to data base systems, ontologyware, and so on.

Most user interfaces are implemented within a single Technological Space (TS), such as Tcl/Tk, Swing, html. This homogeneity does not hold anymore for plastic UI's since re-distribution to different computing devices may require crossing technological spaces. For example, a Java-based UI must be transformed into WML when migrating from a PDA to a WAP-enabled mobile phone.

TS coverage denotes the capacity of the underlying infrastructure to support UI plasticity across technological spaces: *Intra-TS* corresponds to UIs that are implemented and adapted within a single TS. *Inter-TS* corresponds to the situation

where the source UI, which is expressed in a single TS, is transformed into a single distinct target TS. *Multi-TS* is the flexible situation where the source and/or the target user interfaces are expressed in distinct technological spaces. In CamNote, the control panel, when residing on the PC, is implemented with BIGre, a C++ homemade post-WIMP toolkit. It is a Windows-CE toolkit component when it resides on the PDA, whereas the slides display area is kept unchanged as a BIGre widget.

2.3 Existence of a Meta-UI

A *Meta-UI* is the set of functions (along with their user interfaces) that are necessary and sufficient to control and evaluate the state of interactive ambient spaces [11]. This set is *meta-*, since it serves as an umbrella beyond the domain-dependent services that support human activities in an ambient interactive space. They are *User Interface*-oriented since their role is to help users to control and evaluate the state of this space.

A *meta-UI without negotiation* makes observable the state of the adaptation process, but does not allow the user to intervene. This is the case for CamNote: as soon as a PDA enters or leaves the interactive space, UI re-distribution and UI re-molding are automatically applied. The weaving effect of the control panel that expresses its migration between the PC and the PDA is the meta-UI that allows the user to evaluate the evolution of the adaptation process. A *meta-UI incorporates negotiation* when, for example, it cannot make sound decisions between multiple forms of adaptation, or when the user must fully control the outcome of the process.

The balance between system autonomy and too many negotiation steps is an open question. Another issue is the *plasticity of the meta-UI* itself since it lives within an evolving habitat. Thus, the recursive dimension of the meta-UI calls for the definition of a *native bootstrap meta-UI* capable of instantiating the appropriate meta-UI as the system is launched. This is yet another research issue.

2.4 Summary

To summarize, the most demanding case corresponds to the following situation: all aspects of the CARE properties (Complementarity, Assignment, Redundancy, Equivalenece) are supported, from synergistic-complementary multimodality to mono-modal user interfaces. Re-molding and re-distribution are both supported and they both operate at the interactor level while guaranteeing state recovery at the user's action level. They cover all three aspects of the context of use (user, environment, platform). They are able to cross over technological spaces, and they include a plastic meta-UI. Re-molding is able to draw upon multiple modalities that may impact all of the levels of abstraction of an interactive system. And all this can be deployed dynamically.

To address the challenges raised by the development of plastic UI's, we propose a set of principles and approach that are still under development and validation.

3 Principles of our Approach to Plastic UI's

Our principles and approach to the problem of UI plasticity in Ambient Spaces is to bring together MDE and SOA within a unified framework that covers both the development stage and the runtime phase of interactive systems. MDE aims at integrating different technological spaces using models, models transformations and mappings as key mechanisms. SOA defines the appropriate meta-model for a particular class of models: the runtime components. Services are contractually defined behaviors that can be implemented and provided by any software entity for use by any software entity, based on contracts [5]. Contracts range from a non-negotiable level (basic contracts) to a highly dynamically negotiable level (quantitative contracts) [3]. A fundamental characteristic of SOA is that the assembly of service-based software is performed from service descriptors. Discovery and linking between service providers and service customers are made belatedly, just before or at runtime. The flexibility offered by SOA fits our requirements for dynamic UI deployment. In addition, our requirements for fine grained UI components and for re-molding at multiple levels of abstraction are consistent with the modularity and re-usability principles of SOA. The following two principles demonstrate the complementary use of MDE and SOA.

3.1 Principle #1: An interactive system is a graph of models

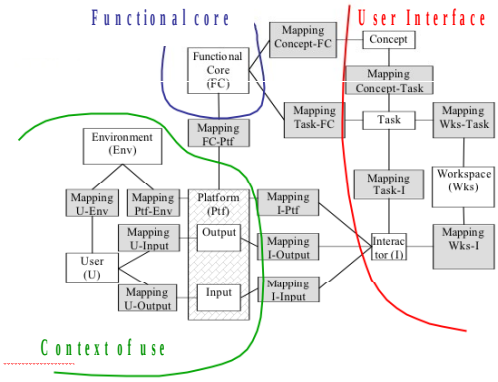


Fig. 2. A UI is a graph of models related by mappings.

An interactive system is a graph of models that expresses and maintains multiple perspectives on the system. As opposed to previous work, an interactive system is not limited to a set of linked pieces of code. As shown in Fig. 2, the models developed at design-time, which convey high-level design decision, are still available at runtime. A UI may include a task model, a concept model, an Abstract UI model (expressed in terms of workspaces), and a Concrete UI model (expressed in terms of interactors) all of them linked by mappings. Tasks and Concepts are mapped to entities of the Functional Core of the interactive system, whereas the Concrete UI interactors are mapped to input and output (I/O) devices of the platform. Mappings between

interactors and I/O devices support the explicit expression of centralized versus distributed UIs.

Transformations and Mappings are models as well. In the conventional model-driven approach to UI generation, transformation rules are diluted within the tool. Consequently, "the connection between specification and final result can be quite difficult to control and to understand" [13, p.13]. In our approach, transformations are promoted as models expressed in ATL [4, 18].

3.2 Principle #2: Close-adaptiveness and Open-adaptiveness Cooperate

A system is close-adaptive when adaptation is self-contained. The system is open-adaptive "if new adaptation plans can be introduced during runtime" [15]. By design, an interactive system has an innate domain of plasticity: it is close-adaptive for the set of contexts of use for which this system can adapt on its own. If it is able to learn adjustments for additional contexts of use, then the domain of plasticity extends dynamically, but this extension relies only on the internal computing capabilities of the system.

In ubiquitous computing, unplanned contexts of use are unavoidable, forcing the system to go beyond its domain of plasticity. If continuity must be guaranteed, then the interactive system must call upon a tier infrastructure that takes over the adaptation process. The role of this infrastructure is to compute the best possible solution by applying new transformations and mappings on the system models and/or by looking for the appropriate services sitting somewhere in the global computing fabric. Fig. 3 shows the functional decomposition of our run-time infrastructure [1].

The platform layer includes the hardware and the legacy operating system(s), which, together, form the habitat of the interactive system. The interactive system layer includes the interactive systems (e.g., CamNote) that users are currently running in the interactive space. The meta-UI is one of them. The structure of an interactive system may be a graph of models as depicted in Fig.2, or a subset of this graph where, for example, the AUI and task model are not maintained explicitly. This MDE approach is combined with the SOA approach of the DMR (Distribution-Migration-Remolding) layer.

The DMR middleware layer supports the context of use and UI adaptation by the way of *services*. The context infrastructure includes a *sensing layer* that generates numeric observables and a *perception layer*, independent of the sensing technology, that provides symbolic observables at the appropriate level of abstraction. The symbolic observables are received by *Observers* that 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 services of the context infrastructure that probe the evolution of the platform. It maintains the platform model shown in Fig. 2. The *physical environment 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 open-adaptation can take over the retargeting process.

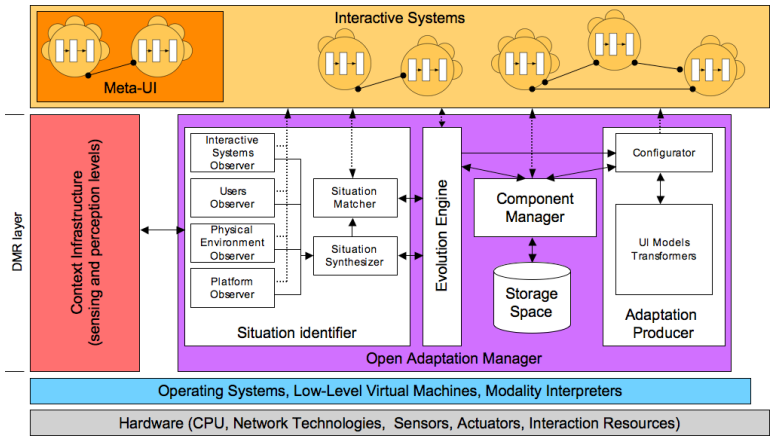


Fig. 3. The overall functional decomposition of a middleware for plastic UI’s.

The *situation synthesizer* computes the current situation from the information provided by the observers. A situation is defined by a set of obervables that satisfies a set of predicates [10]. When the cardinality of the set of observables changes and/or when the predicates do not hold anymore, a new situation is entered. 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 loosing any previous work”.

The *Configurator* executes the plan. If new components are needed, these are retrieved from the *storage space* by the *component manager*. Components of the storage space 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 (state recovery is at the user's action level).

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 *reificators* 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 [6, 16]. Alternatively, the components storage is unable to provide an acceptable solution. In this case, if the graph that depicts the interactive system at run time includes a high-level description such as the task model, then a new CUI can be produced on the fly. The new CUI service may not be as "usable" as expected, but at least, it makes the expected tasks possible: the intended value of the system is preserved.

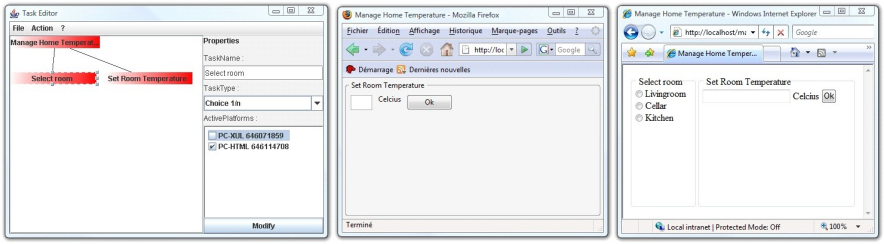


Fig.4 A basic meta-UI making it possible for the user to redistribute the UI.

Fig. 4 shows the flexibility that results from the interplay between modeling an interactive system as a graph of models, the existence of a meta-UI and of UI transformers as services. In this example of a Home Control Heating System (HHCS), the user's task is to set the temperature of the rooms of the home (i.e. the living-room, the cellar and the kitchen). The leftmost window corresponds to a very primitive meta-UI that makes available the task and platform models to the user. The platform model indicates that a PC HTML and a PC XUL are currently available in the home. By selecting a task of the task model then selecting the platform(s) on which the user would appreciate to perform the task, the UI is re-computed and redistributed on the fly, thus ensuring UI consistency. When the user deselects the PC XUL platform for the task "Select room" then the XUL UI is updated accordingly whilst the HTML UI is not changed. The meta-UI is an OSGI service as well as the UI transformers (an embedded ATL transformation engine).

4 Conclusion

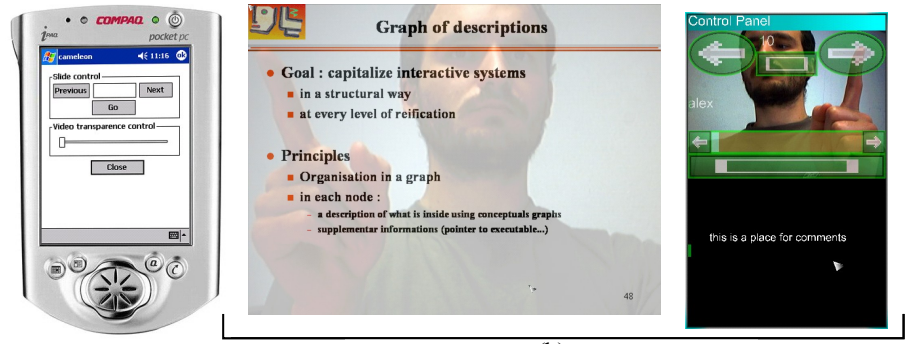
In summary, the problem space of plastic UI covers many dimensions. As a solution space to this complexity, we propose the combined use of MDE with SOA where an interactive system is not simply pure code, but a graph of models related by mappings and transformations, and where general purpose functions such as UI

transformers are encapsulated as services. We shown the promise of our approach with the development of early examples of applications and infrastructure. This work is still under progress.

5 Annex 1

CamNote (for CAMELEON Note) is a slides viewer whose UI is composed of 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, and a control panel to browse the slides and to setup the level of transparency of the pixels mirror. As shown in the picture, the pixels mirror is combined with the slides viewer using alpha-blending. Speakers can point at items on the slide using their finger.

(a)



(b)

Figure a) shows a Pebbles-like configuration where the graphical UI is distributed across the surfaces of a PC and of a PDA. If the PDA disappears, the control panel automatically migrates to the PC screen. Because different resources are now available, the control panel includes different widgets, but also a miniature representation of the speaker's video is now available (cf. Figure b). During the adaptation process, users can see the control panel emerging progressively from the slides viewer so that they can evaluate the progress of the adaptation. The UI, which was distributed on a PC and a PDA is now centralized on an elementary platform. Conversely, if the PDA re-enters the interactive space, the UI automatically switches to the configuration of Figure a) and the control panel disappears from the PC screen by weaving itself into the slides viewer before reappearing on the PDA.

Acknowledgments. This work has been partly supported by Project EMODE (ITEA-if4046) and the NoE SIMILAR- FP6-507609.

References

1. Balme, L., Demeure, A., Barralon, N., Coutaz, J., Calvary, G.: CAMELEON-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces, LNCS Vol. 3295, EUSAI 2004, Springer-Verlag (Publ.), (2004) 291-302
2. Berti, S. & Paternò F. Migratory multimodal interfaces in multidevice environments. In Proc. International Conference on Multimodal Interfaces, ACM Publ., (2005) 92-99
3. Beugnard, A., Jézéquel, J.-M., Plouzeau, N., Watkins, D.: Making components contract aware, IEEE Computer, 13(7), (1999) 38-45
4. Bézivin, J., Dupé, G., Jouault, F., Pitette, G. & Rougui, J. First Experiments with the ATL Transformation Language: Transforming XSLT into Xquery. OOPSLA Workshop, (2003)
5. Bieber, G., Carpenter, J.: Introduction to Service-Oriented Programming (Rev2.1) (2002), http://www.openwings.org/download.html
6. Bouillon, L., Vanderdonckt, J. Retargeting of Web Pages to Other Computing Platforms with VAQUITA, Proceedings of the Ninth Working Conference on Reverse Engineering (WCRE'02), 2002, pp 339
7. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Souchon, N., Bouillon, L., Vanderdonckt, J.: Plasticity of User Interfaces: A Revised Reference Framework. TAMODIA'2002, (2002) 127-134
8. Cockton, G. A development Framework for Value-Centred Design, In ACM Proc. CHI 2005, Late Breaking Results, 2005, pp 1292-1295
9. Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J. & Young, R. Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE properties, Proceedings of the INTERACT'95, , Chapman&Hall Publ., (1995) 115-120
10. Coutaz, J., Crowley, J. , Dobson, S. & Garlan, D.: Context is Key. Communications of the ACM, ACM Publ., 48(3), (2005) 49-53
11. Coutaz, J.: Meta-User Interface for Ambient Spaces, In TAMODIA'06, Hasselt, Belgium, October 2006, Springer LNCS publ., p. 1-15.
12. Kurtev, I., Bézivin, J. & Aksit, M.: Technological Spaces: an Initial Appraisal. CoopIS, DOA'2002 Federated Conferences, Industrial track, Irvine (2002)
13. Myers, B., Hudson, S.E. & Pausch, R.: Past, Present, and Future of User Interface Software Tools. Transactions on Computer-Human Interaction (TOCHI), ACM Publ., Vol 7(1), (2000) 3-28
14. Njike, H., Artières, T., Gallinari, P., Blanchard, J., Letellier, G. Automatic learning of domain model for personalized hypermedia applications, International Joint Conference on Artificial Intelligence, IJCA, Edinburg, Scotland, 2005, pp 1624
15. Oreizy, P., Gorlick, M., Taylor, R., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D., Wolf, A.: An Architecture-Based Approach to Self-Adaptive Software", IEEE Intelligent Systems, May/June (1999) 54-62
16. Paganelli, L., Paternò, F. Automatic Reconstruction of the Underlying Interaction Design of Web Applications, Proceedings Fourteenth International Conference on Software Engineering and Knowledge Engineering, ACM Press, Ischia, July 2002, pp 439-445
17. Sheshagiri, M., Sadeh, N., Gandon, F. Using Semantic Web Services for Context-Aware Mobile Applications, In Proceedings of ACM MobiSys2004 Workshop on Context Awareness, Boston, Massachusetts, USA, June 2004
18. Sottet, J.-S., Calvary, G., Favre, J.-M.: Towards Model Driven Engineering of Plastic User Interfaces. International workshop on Model Driven Development of Advanced User Interfaces (MDDAUI), MoDELS 05 (2005)