Plastic User Interfaces: Designing for Change

Montserrat Sendín, Jesús Lorés
Computer Science Department
University of Lleida
69, Jaume II St., 25001- Lleida, SPAIN
+34 973 70 2 700
{msendin, jesus}@eup.udl.es

ABSTRACT
Everybody knows that mobile computing provides us with tremendous versatility. But this versatility also increases in a great deal the complexity in the design of User Interfaces (UI). It is obvious the necessity of an architectural framework that provides dynamic adaptation.
We propose a reflective architecture to manage the system to adjust its own behaviour according to certain runtime conditions, related to the context of use. The benefits of reflection are, apart from dynamic adaptation, transparency and reusability. We also present the underlying plastic UIs development framework, inspired in the model-based approach . Its focus of attention is also to solve contextual changes, as one of the most important lacks detected in literature.

Keywords
model-based approach, context of use, reflection, plasticity

INTRODUCTION

Today technology allows users to move about with computing power and network resources at hand. Computers are shrinking while the bandwidth of wireless communications keeps increasing. These changes have increasingly enabled access to information “anytime and anywhere”, making computing possible in multiple and varied *contexts of use* (set of environment parameters that describe a particular context where the interaction takes place by a determined user). Nevertheless, *runtime conditions* related not only to resources constraints (bandwidth, server availability, physical resources, etc.), but also other related to the user (user mobility, user’s changing needs, tasks to be developed, profile and current situation), and even related to the environment (day of the week, hour, weather conditions, etc.), are volatile and require sophisticated adaptive capabilities that today are still challenging. The adaptation to this continuous and diverse variability must be solved as automatically as possible, and this raises an important challenge. We refer to the *context-awareness* issue. [1] offers a detailed survey.

Designing this kind of systems becomes a complex task, due to a number so high of concerns to consider. This involves a lot of decisions about how these runtime conditions must be modelled, as well as the way in which they interact. Furthermore, adaptive capabilities should be incremental. It is also desirable for the adaptive capabilities and the system’s core functionality to be handled orthogonally, so that they can evolve individually and promote system’s flexibility. Besides, the adaptation mechanism should be transparent.
Reflection techniques arise as a promising tool to develop context-aware systems, because they provide all of previous requirements. A reflection system has the capability to reflect about its own status and behavior and adapt those if the need arises. This leads to the separation of the UI component into an abstract representation of requirements and constraints, and a set of concrete sub-components which are assembled into a specific interface instantiation according to the current context in which the system is used. We can assert that reflective computation provides an architecture for the adaptation. But using reflection we obtain a set of additional benefits. As the core application abstracts from context constraints, it becomes easier to design and implement. Finally, separation of concerns allows to obtain a reusable context representation and adaptation mechanism. For example, we could think in a tourism application, as is presented in [10], and also, for example, in a telemedicine application.
But apart from the context-awareness issue, it is necessary to realize that such a multiplicity of contexts of use requires making available a specific UI suitable for each case. This imposes another important challenge if we want to solve it without falling into an extremely repetitive process, and preserving at the same time consistence and usability.
These considerations motivated the development of *generic methodologies* (*model-based approach* [7] and *appliance-independent XML-based languages* [6]), arisen with the aim of making this process flexible. The idea consists of specifying a unique *generic UI*[1], flexible enough to cope with multiple variations, producing so as many of UIs as necessary. The goal is to guarantee usability continuity under any variation, while minimising development cost. This capacity of adaptation from a same generic UI to different contexts of use is called *plasticity property* [12].

What we intend is to develop a plastic UIs development framework as a support of systematic and dynamic development, and whose focus of attention is solving the

[1] UI whose aspects can vary in different devices, while its functionality prevails in all of them.

anticipation to contextual changes, as one of the most important lacks detected in literature.

MODEL-BASED APPROACH: STATE OF THE ART

The main idea of this kind of techniques is, on one hand, the fact that all of the relevant aspects of the UI are explicitly formalised and represented in declarative models that get together all the different requirements of each context of use (the *interface model*[2]), storing that way the conceptual representation of the interface. On the other hand, this kind of techniques also provides methods and tools that exploit these models for supporting the systematic development of the interface. The assemblage of the interface model with the underlying development tools is what is called *MB technique* [9].

MB techniques provide a lot of benefits. We can remark these ones as the most remarkable:

- Provide a more abstract description of UI than other UI development tools.
- Facilitate the creation of methods to design and implement UI in a systematic way and provide infrastructure to automate tasks related to UI design.
- Provide a comprehensive support of the whole system life-cycle.
- They are a user-centred design methodology.

There exist a great variety of examples. Even we can distinguish between a first and a second generation of MB techniques. See [8] to look up a complete overview of some of the best-known MB techniques.

Particularly we have revised in depth the MB techniques developed for three members of the RedWhale: Eisenstein, Vanderdonckt and Puerta [2], the method for *Universal Design of UIs* in [3], a framework for supporting plasticity [12], as well as the TERESA tool [6]. In general, the techniques proposed until now are substantially static. This means that situations provoked by contextual changes are not enough anticipated. It will be better a more dynamic solution.

In conclusion, although there are commercial products that use this kind of tools, there exist aspects that must be studied in order to increase their acceptation. In general, we can remark this set of problems and shortcomings:

- Complexity of the models and their notations.
- Multiple and meaningful differences (no consensus) in range, nature and notation of supported models, which make difficult the comparison and reutilization of models. It would be beneficial to dispose some standard notation.
- Difficulty to model the relationships between models (the *mapping* problem).
- The problem of post-editing refinements.
- They mostly support the generation of form-based UIs only (limited set of interactors, and also very simple).
- The problem of integrating the different UIs with their underlying application.

But, as far as we are conserned, the problems that we consider more important are the next ones:

- The problem of flexibility in content adaptation and coherence, that remains unsolved .
- The fact that they are in general substantially static, leaving without solving the anticipation to contextual changes, as it occurs, for example, in ARTStudio (Adaptation by Reification and Translation). This is the problem in which we are mainly focused.
- The lack of semantic information inside the models.

Definitely, the MB-UIDE should improve for addressing the problem of plasticity. We can assert that they are still challenging.

In our opinion, the set of models taken into account is quite limited. In general, they only consider tasks, users and platform models, leaving without modelling the contextual aspects, except in punctual cases. As a consequence of that, the anticipation to contextual changes rests still without solving. In our opinion, the context model has also to be considered and appropriately related to the rest of models, taking part in the process of constructing plastic UIs.

It is worthy to say that despite the apparent correlation between the user model and the context model, the utilisation of user modelling techniques within the domain of context-aware computing is a relatively unexploited research area.

OUR PROPOSAL

The Reflective Architecture

Under an object-oriented reflective architecture view [5, 11, 14], a system is considered integrated by two parts: the application part and the reflective part, which is capable of reasoning about and acting upon itself. These parts reside in two different levels: the base level and the meta level, respectively. The components related to the functionality of the application are represented at the base level, and they are manipulated by the meta level. The base level has no knowledge about the existence of the other one. This feature lets designers to isolate the behaviour of the base level form the assigned orthogonal properties in the meta level.

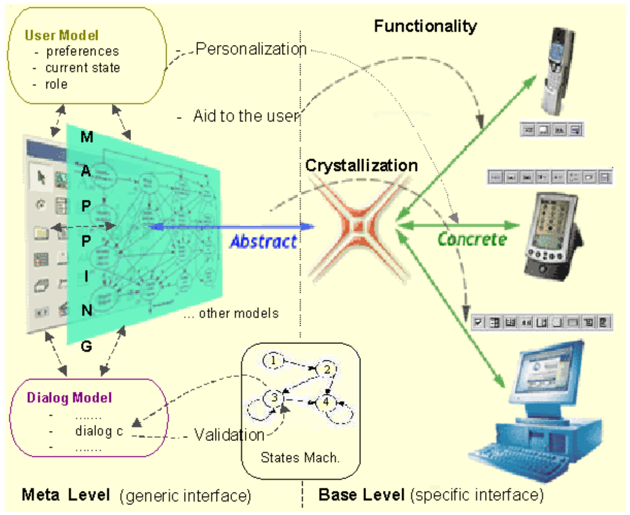
The meta level provides a representation of the behaviour of the system and internal structure. It is commonly called *auto-representation*. This auto-representation is amenable to inspection and adaptation (*introspection property*), and also is *causally-connected* to the underlying behaviour it describes. This means that changes made to the self-representation are immediately mirrored in the underlying system's actual state and behaviour, and vice-versa (*reflection property*).

[2] Formal, declarative and implementation-neutral description of the UI, that should be expressed by a modeling language.

Our objective consists of developing a system capable of adapting the UI according to a set of conditions related to the context of use. These conditions will be modelled and treated isolatedly in the meta level, with the aim of providing the required adaptation. The part destined to the functionality will reside in the base level of the architecture. Both of them will be independent each other. That way, in the base level, following the abstraction principle, the system work without any conception of interface. This will be taken on in the meta level at run-time, fixing which concrete interface components will represent the functionality described by the abstract components, depending on the contextual situation (final device, the user's profile and other runtime conditions). This will be the main responsibility of the meta level and it will be realised in a transparent way to the user.

Moreover, following the *component-oriented programming approach*, the base level can be formed, independently of his functionality, by a control structure similar to a *states machine*, where each state is associated to a dialog, identified in the design phase. The transition from one state to another one will not be realised until the user provides the necessary information and this is validated. Again this validation task is to be done by the meta level. Figure 1 represents that idea.

Fig. 1. Reflective architecture and distribution of responsibilities



Plastic UIs Development Framework

The models that we consider relevant from the point of view of this kind of UIs are the next ones: user model (UM), task model (TM), domain model (DM), dialogue model (DgM), presentation model (PM), platform model (PltM) –explicit expression of the target platforms in terms of quantified physical resources-, spatial model (SM) –the detailed spatial model from the real world-, and finally the contextual model (CM) -to take into account daily aspects, which also can influence in the adaptation.

Apart from these models, there are other specifications that also take part in the process: the *Abstract User Interface* (AUI) and the *Concrete User Interface* (CUI). We define the AUI as an abstract specification of the layout of the resulting interface (set of abstract interactors), as a static structure, as well as a description about how the UI evolves over the time. It is high-level and appliance-independent. We define the CUI as a concrete instance of an AUI, low-level and appliance-dependent. Lets go to outline the general description of our framework.

- Composed of two sequential phases called *Abstract Rendering Process* (ARP) and *Concrete Rendering Process* (CRP) respectively, in which vary the set of models to take part. The first stage is in charge of obtaining the AUI. The models that intervene are the next ones: SM, TM, DM, UM and DgM. The second stage manages the selection of the set of final interactors, which reside in the PM, according to all the contextual information represented in the next models: UM, SM, PltM and CM, and also ruled by the DgM. More concretely, it is in charge of translating each abstract interface object in the AUI to a concrete interface object according with the current situation. As a result, this stage obtains the expected CUI, resulting from the restrictions propagation.
- We propose to use model repositories. This allows each model to populate a common area with the specific concepts it is responsible for capturing. We use a model repository for each rendering process, making possible to share concepts between the models.
- Equally, we consider necessary to use two groups of mapping rules, one for phase, to manage the relations in each group.
- There also intervene some ergonomic heuristics, style guidelines and usability patterns [13] in the second phase to manage the transformation from Abstract Interaction Objects (AIO) to Concrete Interaction Objects (CIO), according to some environmental circumstances and to preserve usability.

This model corresponds to a *shared model approach* that allows informing the other models of any change to any concept produced in the UI, providing so a propagation mechanism. The lack of a mechanism to propagate changes is one of the relevant limitations we have detected in the model-based tools we have analyzed.

These ideas have been inspired in the approach used in the Teallach system [4]. Figure 2 shows the sketch of our plastic UIs development framework, depicting all the relations among the models.

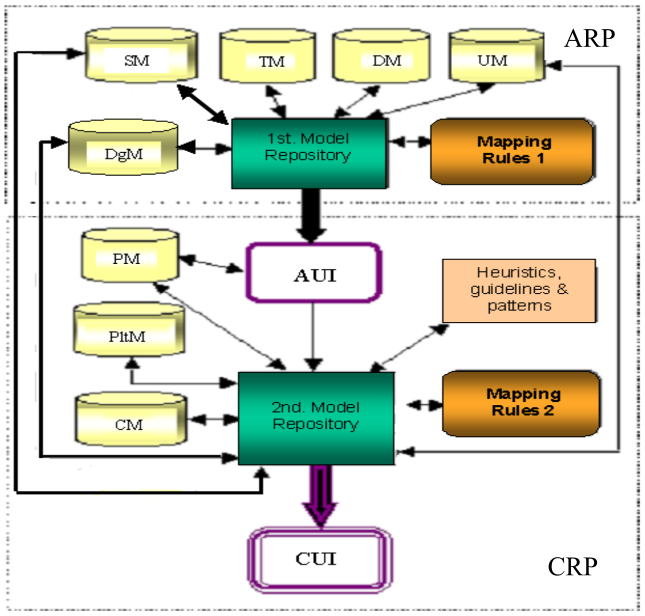


Fig. 2. Sketch of our development framework

CONCLUSIONS

This paper sets up two important challenges in the development of mobile applications: the context awareness and the systematic development of plastic UIs. We could snap the present situation that way: there not exists a UI development tool with a enough high abstraction level in the design phase, leaving unsolved a lot of problems; there not exists a wholly appliance-independent language that neither integrate all of the content adaptability aspects, nor that take into account all the research work in the field. It is obvious the necessity of developing an architectural framework that adjust this diversity.

Though the benefits of incorporating a context-sensitive component into a mobile computer application may be immense, the actual practicalities of doing so present some difficult design and implementation problems. We propose reflection as the mechanism to solve the context awareness. Reflection offers a lot of advantages. One of the most important is that thanks to the separation of concerns, the developer only has to focus on modelling the functionality of the application, without considering the interface, obtaining so a reusable adaptation mechanism.

This paper presents a reflective architecture and the underlying framework to develop plastic UIs "on the fly". These ideas, arisen from a deep analysis of the existing techniques and inspired in the model-based approach, propose a solution to some shortcomings detected.

The utilisation of abstract models –neutrals regarding the platform and the user's typology, and user-centred- to describe the UI makes substantially easier the development of consistent and usable UIs to mobile devices. This is that way due to the exploitation of the aforementioned models by means of a model-based technique provides an automated support that allows designers to surpass the challenges posed by mobile computing.

REFERENCES

1. Chen, G. and Kotz, D.: A survey of context-aware mobile computing research. Technical Report TR2000-381, Computer Science Department, Dartmouth College (Hanover, New Hampshire, November 2000)

2. Eisenstein, J., Vanderdonckt, J., Puerta, A.: Adapting to Mobile Context with User-Interface Modeling. Workshop on Mobile Computing Systems and Application. Monterey (2000)

3. Furtado, E., Vasco, J., Bezerra, W., William, D., da Silva, L., Limbourg, Q., Vanderdonckt, J.: An Ontology-Based Method for Universal Design of User Interfaces. Workshop on Multiple User Interfaces over the Internet: Engineering and Applications Trends (2001)

4. Griffiths, T., Barclay, P., McKirdy, J., Paton, N., Gray, P., Kennedy, J., Cooper, R., Goble, C., West, A., and Smyth, M. Teallach: A Model-Based User Interface Development Environment for Object Databases. Interacting with Computers, Vol. 14, No. 1 (December 2001), 31-68.

5. Maes, P.: Concepts and Experiments in Computional Reflection. Proc. of the 2nd OOPSLA'87 (1987) 147-156

6. Mori, G., Paterno, F., Santoro, C. Tool support for designing nomadic applications.

7. Paternò, F.: Model-Based Design and Evaluation of Interactive Applications. Springer-Verlag, London (2000)

8. Pinheiro, P.: The Unified Modeling Language for Interactive Applications. http://www.cs.man.ac.uk/img/umli/links.html

9. Pinheiro, P. User interface declarative models and development environments: a survey, in Proceedings of DSV-IS'2000, P. Palanque and F. Paterno (Ed.)

10. Sendín, M., Montero, F., López, V., and Lorés, J:. Towards a framework to develop plastic user interfaces, in *Proceeding of Mobile HCI'03* (Udine, Sep. 2003), Springer Verlag, 428-433.

11. Smith, B.C.: Reflection and Semantics in Lisp. Proc. of ACM Symposium on Principles of Programming Languages (1984) 23-35

12. Thevenin, D., and Coutaz, J.: Plasticity of User Interfaces: Framework and Research Agenda, in Proceedings of Interact'99, (Edinburgh, 1999), 110-117.

13. Tidwell, J.: UI Patterns and Techniques. http://time-tripper.com/uipatterns (2002).

14. Zimmerman, C.: Advances in Object-Oriented Metalevel Architectures and Reflection. *CRC Press*, Inc., Boca Raton, Florida 33431 (1996)