Requirements and models for next generation UI languages

Alexandre Demeure
University of Grenoble, LIG
385, rue de la bibliothèque B.P. 53
38041 Grenoble Cedex 9, France
Alexandre.Demeure@imag.fr

Gaëlle Calvary
University of Grenoble, LIG
385, rue de la bibliothèque B.P. 53
38041 Grenoble Cedex 9, France
Gaelle.Calvary@imag.fr

CHI 2007, April 28 – May 3, 2007, San Jose, USA
ACM 1-xxxxxxxxxxxxxxxxxx.

Abstract
In this paper we argue that concrete User Interfaces (UI) languages are not suitable for adaptation. In addition, we point out the fact that the quality of tailored UIs is far better than the quality of automatically generated UIs. Therefore we propose to capitalize human designed UIs in a structured knowledge base as promoted by Service Oriented Approaches. The base aims at supporting designers and/or automatic UI generation algorithms in retrieving UI descriptions both at design time and runtime.

Keywords
Plastic User Interface (UI), UI Adaptation, Model Driven Engineering, Service Oriented Architecture, UI Description Language, Tailored UIs.

ACM Classification Keywords
H5.2. Information interfaces and presentation (e.g., HCI): User Interfaces – User Interface Management Systems (UIMS).

Introduction
With the rise of ubiquitous computing, a lot of work has been done about User Interface (UI) Description Languages (UIDL). The main goal was to enable designers to describe UIs at a high level of abstraction

and then automatically generate the final UI for different platforms. Later on, the CAMELEON reference framework [4] made explicit four levels of abstraction for describing a UI: concepts and tasks (C&T), Abstract UI (AUI), Concrete UI (CUI) and Final UI (FUI).

One problem in UI generation is that only WIMP languages or WIMP toolkits (SWING, XHTML, etc.) are targeted. As a result, UIs are of a poor quality and deprived of any non WIMP interaction style. This is probably due to the fact that WIMP toolkits are standard de facto.

The right model at the right place

Whilst task languages are almost well formalized, providing just the required information, the same can not be said for current CUI languages. One of the problems is that they just enumerate "classical" WIMP widgets. These widgets implicitly mix several levels of abstraction. For instance, a button can be described at the CUI level (a clickable box with a text inside) as well as at the task level (it supports the "activation" task). This is probably due to the time when no task model was used to model a UI. It is no more the case but leads to two main drawbacks: each widget is associated with a particular task and symmetrically each task has a finite and frozen set of possible presentations. Such rigid associations are the result of years of practice. It is valuable within the WIMP assertion (one user, one screen, one keyboard, one mouse …) but becomes inadequate in ubiquitous computing. The same applies to the look and feel of the widgets. The way of interacting with a widget is most of the time hard-coded in the widget itself, disabling the possibility for the user to use other interaction modalities such as gesture or vocal recognition.

Model Driven Engineering (MDE) [3] shows that each level of abstraction should be described along an appropriate specific language. These languages should focus on their abstraction level (C&T, AUI, CUI, FUI) without introducing artificial dependencies with other abstraction levels. Mappings are devoted to such relationships.

In practice, these comments apply to CUI languages or toolkits. However, some works separate rendering from interaction [1],[2] and tend to overcome classical widgets [4]. The trend (at least for graphical UIs) seems to be the definition and organization of drawing primitives among a scene graph. "Widgets" are built as assemblies of primitive nodes [[7],[4]]. This makes easier the design of nice looking UIs and as a result favors the exploration of new designs. Other approaches [8] explore arbitrary abstract widgets that can be mapped on concrete and possibly non standard widgets on the fly, thus enhancing the diversity of presentations.

Generated versus tailored UIs

When considering quality in use, tailored UIs (i.e., UIs designed by a human) are far away from what a program can automatically generate (Figure 1). This is partially due to the fact that computers know almost nothing about notions such as "beauty" yet. Figure 1-B shows two WinAMP skins. Compared to Figure 1-A, a lot of graphical artefacts have been added: they do not correspond to functional requirements of an audio player (light effect, rounded forms, buttons layouts, etc.). They improve the UI quality in use. Today, it is still impossible to integrate them automatically.

This leads to the following conclusion: "if we can not do it by ourselves and if there is no hope for the near future then why not reusing human pre-built tailored UIs?". In Service Oriented Approaches (SOA), a service can be achieved by combining smaller services provided by other people. In the same way, the UI of a "large" system could be composed of smaller ones tailored by designers for a particular context. Automatically composing tailored UIs would provide great benefits but this requires that each UI is described in a processable way.

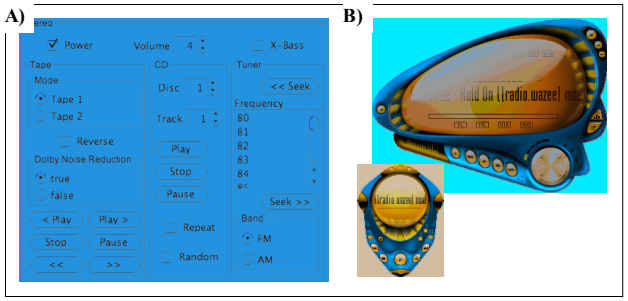

Figure 1. A) UI of an audio player generated by the SUPPLE system [6]. B) Two tailored UIs of the WinAMP audio player.

Besides the description of UIs, another issue is where to store and find tailored UIs.

Capitalizing knowledge

Capitalizing and giving access to services is a key point of SOA. SOA promotes service brokers (Figure 2). In the same way, capitalizing and giving access to UI descriptions or implementations should be a key point of UI generation (automatic, semi-automatic or manual).

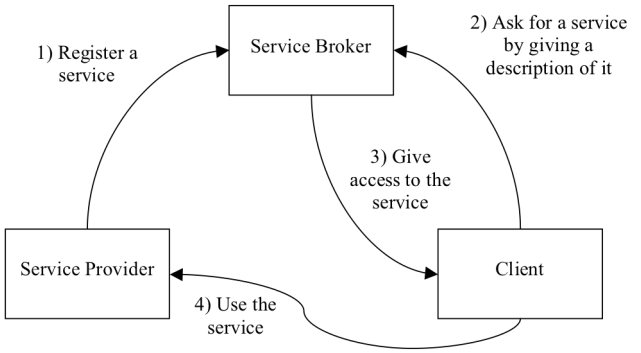

Figure 2. Global view of SOA.

The difference between SOA and Human Computer Interaction (HCI) is that services descriptions are mainly functional in SOA, whist HCI requires extra-functional descriptions. Some works, like in [5], explore the possibility of using a semantic network to classify and retrieve UI models or implementations both at design time and runtime. Each node corresponds to the model of an interactive system. The model can be done at any level of abstraction (C&T, AUI, CUI, FUI). Each edge of the network corresponds to a relationship between the two corresponding models. Classical relationships are inheritance, specialisation, extension, restriction, composition, etc. The structure of the network provides a support for solving "plasticity questions" such as: "Is there a pre-computed UI for supporting this task on this platform or on this context of use?". The question is translated in terms of a logical path in the semantic network starting from the node describing the task.

Conclusion

MDE for HCI clearly separates the UI models according to their level of abstraction. Whist task languages are almost well formalized and general enough, CUI languages are mostly WIMP oriented. We need CUI languages suitable for each modality. At least, we think it is possible to define a GUI language that covers post WIMP interaction. This language should describe stuff like geometry, colors, textures, behavior, rich inputs, etc. As automatically generating such descriptions may be difficult and automatically generated UIs are mostly of a poor quality, we believe in tailored UIs, capitalized in a structure (e.g. a semantic network), and retrieved at design time as well as at runtime.

Bibliography

[1] Appert, C. and Beaudouin-Lafon, M. 2006. SwingStates: adding state machines to the swing toolkit. In *Proceedings of the 19th Annual ACM Symposium on User interface Software and Technology*. UIST '06. ACM Press, New York, NY, 319-322.

[2] Blanch, R. and Beaudouin-Lafon, M. 2006. Programming rich interactions using the hierarchical state machine toolkit. In *Proceedings of the Working Conference on Advanced Visual interfaces*. AVI '06. ACM Press, New York, NY, 51-58.

[3] Calvary G., Coutaz J., Thevenin D., Limbourg Q., Bouillon L., Vanderdonckt J., *A Unifying Reference Framework for Multi-Target User Interfaces*, Interacting With Computers, Vol. 15/3, pp 289-308, 2003.

[4] Chatty, S., Sire, S., Vinot, J., Lecoanet, P., Lemort, A., and Mertz, C. 2004. Revisiting visual interface programming: creating GUI tools for designers and programmers. In *Proceedings of the 17th Annual ACM Symposium on User interface Software and Technology*. UIST '04. ACM Press, New York, NY, 267-276.

[5] Demeure A., Calvary G., Coutaz J., Vanderdonckt J., *The COMETs Inspector: Towards Run Time Plasticity Control Based on a Semantic Network*, TAMODIA'2006.

[6] Gajos, K. and Weld, D. S. 2005. Preference elicitation for interface optimization. In *Proceedings of the 18th Annual ACM Symposium on User interface Software and Technology*. UIST '05. ACM Press, New York, NY, 173-182.

[7] Lecolinet E., *A Brick Construction Game Model for Creating Graphical User Interfaces: The Ubit Toolkit*. In Proc. INTERACT'99, 1999

[8] UIML. http://www.uiml.org.