
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
Gaëlle.Calvary@imag.fr

Abstract

In this paper we explain why concrete User Interfaces (UI) languages are too high level to be flexible enough for adaptation. Moreover, we note that the quality of UIs produced by human designers is far away from quality of automatically generated UI. Therefore we propose to classify human designed UI in a knowledge base inspired by Service Oriented Approaches. This base aims to allow designer or even automatic UI generation algorithms to retrieve UI descriptions both at run time and design time.

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 give conceptors the opportunity to describe the UI at a high

level of abstraction and then generate automatically the “real” UI for different platforms. CAMELEON framework [4] explicit the relevant level of abstraction at which a UI can be described: concepts and tasks (C&T), Abstract UI (AUI), Concrete UI (CUI) and Final UI (FUI).

The problem is that generation is done using WIMP languages or WIMP toolkits (SWING, XHTML, etc.) which result in poor quality UI and exclude any non WIMP interaction style. The main reason for this state of fact is probably that WIMP toolkits are de facto standard.

The right model at the right place

While task languages are well formalized, providing neither more nor less than information needed, the same can not be said about current CUI languages. One of the problems with these languages is that they enumerate “classical” WIMP widgets. These widgets combine implicitly several level of abstraction. For example, a button can be described at a CUI level (a clickable box with a text inside) as at a task level (it support the “activation” task). This state of fact come from the time when there were no task models used to model the UI. It is no more true and leads to two main drawbacks: each widget is associated with a particular task and symmetrically each task only has a finite and frozen set of possible presentations. This association is the result of years of practice. It is valuable within the WIMP assertion (one user, one screen, one keyboard, one mouse ...) but may be inadequate in other context. The same remark can be done about how the widget is rendered and how it is possible to interact with it. The way to interact with a widget is most of the time hard-coded in the widget itself, disabling the possibility for

the user to use other interaction paradigms such as gesture or vocal recognition.

MDE approaches [3] show us that each level of abstraction should be described using an appropriate language. These languages should only describe information related to their abstraction level (C&T, AUI, CUI, FUI) and should not introduce artificial dependencies with other abstraction level. Instead, mappings should be used to express relationships between descriptions of different level of abstraction.

In practice, these remarks apply to CUI languages or toolkits. However, some works separate rendering from interaction [1],[2] and tend to overcome classical widgets [4]. The trend seems to be to propose, at least for GUI, sets of drawing primitives organized in a scene graph, “widget” are reconstruct as an assemblies of primitives nodes [[7],[4]]. These approaches allow designers to produce more easily nice looking UIs and avoid closing the possible presentations of a given task. Other approaches [8] tend to propose arbitrary abstract widgets that can be mapped on the fly to concrete and possibly non standard ones, thus enhancing diversity of presentations.

Generated versus tailored UIs

If we consider quality of use, tailored UI (UI designed by a human) are far away from what a program can automatically generate (Figure 1). These can be explained by the fact that, for now, computers know almost nothing about notions such as “beauty”. Figure 1-B show some WinAMP skins, we can see that compared to Figure 1-A, a lot graphical artefact are added that do not correspond to any functional requirement of an audio player (light effect, rounded

forms, buttons layouts, etc.). These artefacts are produce by human designer to improve the quality of use and are for now impossible to be automatically generated.

This leads us to the following conclusion: "if we can not do it by ourselves and there is no hope to it in a close future, why not using human pre-build, tailored, UI?". In SOA, a service can be achieved by combination of smaller services done by other peoples. By symmetry, the UI of a "big" system could be composed by smaller UI tailored by designers for a particular context. Integrating these tailored UIs automatically would provide great benefits provided that it will be possible to have a processable description of them.

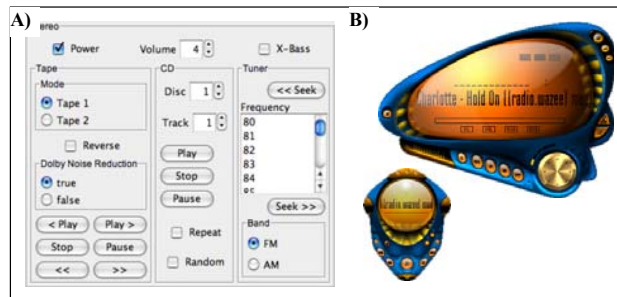


Figure 1. A) UI of an audio player generated with SUPPLE system [6]. B) Two UI of the WinAMP audio player.

The problem is now to know where these tailored UIs can be found.

Capitalizing knowledge

Capitalizing and giving access to services is a key point of SOA. To achieve it, SOA use service brokers (Figure 2). The same way, capitalizing and giving access to UIs

descriptions or implementations should be a key point for UI generation (automatic, semi-automatic or manual).

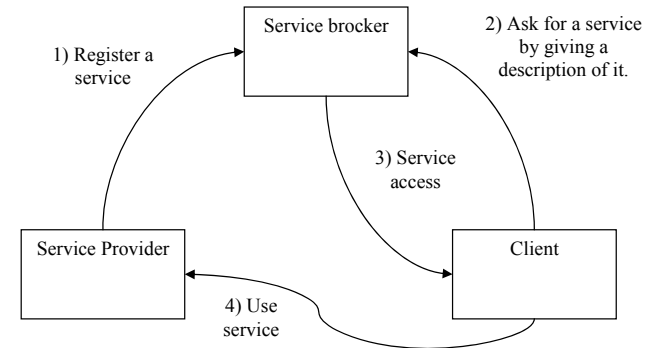


Figure 2. Global view of SOA.

The difference is that services descriptions are mainly functional, while we need UI 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 and run-time. Each node corresponds to the model of an interactive system. This model can be done at any level of abstraction (C&T, AUI, CUI, FUI). Each edge of this network corresponds to a relationship that can be established between two models. Classical relationships are inheritance, specialisation, extension, restriction, composition, etc. The structure of this semantic network provides a support to solves some "plasticity question" such as: "Is there a UIs pre computed for this task that is tailored for this platform or, more generally speaking, this context of use?". The question can be translated in term of a logical path in the graph starting from the node describing the task.

Conclusion

IDM for HCI clearly separates the UI models depending on their level of abstraction. While task languages are well formalized and general enough, CUI languages are mostly WIMP oriented. We need CUI languages adapted to each modality. At least, I think it is possible to define a GUI language that enables post WIMP interaction. This language should describe stuff like geometry, colors, textures, behavior, rich inputs, etc. Generating automatically such description may be difficult. Moreover, automatically generated UI are poor quality in general. That's way we should take into account tailored UI by capitalizing them in a structure (e.g. a semantic network) allowing to retrieve them both at design time and 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., Vanderdonck 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., Vanderdonck 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>.