
The Many Faces of Plastic User Interfaces

Gaëlle Calvary

Université Joseph Fourier
Grenoble Informatics Lab.
BP 53
38041 Grenoble Cedex 9,
France
gaelle.calvary@imag.fr

Joëlle Coutaz

Université Joseph Fourier
Grenoble Informatics Lab.
BP 53
38041 Grenoble Cedex 9,
France
joelle.coutaz@imag.fr

Lionel Balme

Université Joseph Fourier
Grenoble Informatics Lab.
BP 53
38041 Grenoble Cedex 9, France
lionel.balme@imag.fr

Alexandre Demeure

Université Joseph Fourier
Grenoble Informatics Lab.
BP 53
38041 Grenoble Cedex 9, France
alexandre.demeure@imag.fr

Jean-Sebastien Sottet

Université Joseph Fourier
Grenoble Informatics Lab.
BP 53
38041 Grenoble Cedex 9, France
jean-sebastien.sottet@imag.fr

Abstract

In this paper, we discuss the problem of UI adaptation to the context of use. To address this problem, we propose to mix declarative languages as promoted in Model Driven Engineering (MDE) with a “code-centric” approach where pieces of code are encapsulated as service-oriented components (SOA), all of this within a unified software framework that blurs the distinction between the development stage and the runtime phase. By doing so, we support UI adaptation where conventional WIMP parts of a user interface can be (re)generated from declarative descriptions at the level of abstraction decided by the designer, and linked dynamically with hand-coded parts that correspond to the post-WIMP portions of the UI whose interaction nuances are too complex to be expressed with a UIDL. We have experienced different methods for mixing MDE with SOA at multiple levels of granularity.

Keywords

Plastic User Interface. User Interface adaptation, context-sensitive user interface, Model Driven Engineering, Service Oriented Architecture, User Interface Description Language.

ACM Classification Keywords

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

Introduction

With the move to ubiquitous computing, it is increasingly important that user interfaces (UI) be adaptive or adaptable to the context of use (user, platform, physical and social environment) while preserving human-centered values [3]. We call this “UI plasticity”. From the software perspective, UI plasticity goes far beyond UI portability and UI translation.

As discussed in [4], the problem space of plastic UI is complex: clearly, it covers UI re-molding, which consists in reshaping all (or parts) of a particular UI to fit the constraints imposed by the context of use. It also includes UI re-distribution (i.e. migration) of all (or parts) of a UI across the resources that are currently available in the interactive space. UI plasticity may affect all of the levels of abstraction of an interactive system, from the cosmetic surface level re-arrangements to deep re-organizations at the functional core and task levels. When appropriate, UI re-molding may be concerned by all aspects of the CARE properties, from synergistic-complementary multimodality (as in “put-that there”) and post-WIMP UI’s, to mono-modal GUI. Re-molding and re-distribution should be able to operate at any level of granularity from the interactor level to the whole UI while guaranteeing state recovery at the user’s action level. Because we are living in a highly heterogeneous world, we need to support multiple technological

spaces¹ simultaneously such that a particular UI may be a mix of, say, Tcl/Tk, Swing, and XUL. And all of this, should be deployed dynamically under the appropriate human control by the way of a meta-UI [4].

Observations

Our approach to the problem of UI plasticity is based on the following observations:

(1) The software engineering community of HCI has developed a refinement process that now serves as a reference model for many tools and methods: from a task model, an abstract UI (AUI) is derived, and from there, the Concrete UI (CUI) and the Final UI (FUI) are produced for a particular targeted context of use. The process is sound but cannot cope with ambient computing where task arrangement may be highly opportunistic and unpredictable.

(2) 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. In particular, the links between the FUI and its original task model are lost. As a result, it is very hard to re-mold a particular UI beyond the cosmetic surface.

(3) Pure automatic UI generation is appropriate for simple (not to say simplistic, “fast-food”) UI’s. As

¹ “A technological space is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities.” [5]

mentioned above, the nuances imposed by high-quality multi-modal UI's and post-WIMP UI's, call for powerful specification whose complexity might be as high as programming the FUI directly with the appropriate toolkit. In addition, conventional UI generation tools are based on a single target toolkit. As a result, they are unable to cross multiple technological spaces.

(4) Software adaptation has been addressed using many approaches over the years, including Machine Learning, Model-Driven Engineering (MDE), and service-oriented components. These paradigms have been developed in isolation and without paying attention to UI-specific requirements. Typically, a "sloppy" dynamic reconfiguration at the middleware level is good enough if it preserves system autonomy. It is not "observable" to the end-user whereas UI re-molding is! Thus, UI re-molding adds extra constraints such as making explicit the transition between the source and the target UI's so that, according to Norman, the end-user can evaluate the new state.

Based on these observations, we propose the following key principles that we have put into practice using a combination of MDE and SOA. The exploration of Machine Learning is under way.

Three Principles for UI Plasticity

Principle #1: Close-adaptiveness must cooperate with open-adaptiveness. 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/component can adapt on its own. In ubiquitous computing, unplanned contexts of use are unavoidable, forcing the system to go beyond its domain of plasticity. Then the interactive system must be open-

adaptive so that a tier infrastructure can take over the adaptation process. The functional decomposition of such an infrastructure is described in [1].

Principle #2: An interactive system is a set of graphs of models that express different aspects of the system at multiple levels of abstraction. These models are related by mappings and transformations, which in turn, are models as well. As a result, an interactive system is not limited to a set of linked pieces of code. The models developed at design-time, which convey high-level design decision, are still available at runtime for performing rationale deep adaptation. In addition, considering transformations and mappings as models is proving very effective for controlling the adaptation process [6].

Principle #3: By analogy with the slinky meta-model of Arch, increasing principle #1 allows you to decrease principle #2 and vice-versa. At one extreme, the interactive system may exist as one single task model linked to one single AUI graph, linked to a single CUI graph, etc. This application of Principle#1 does not indeed leave much flexibility to cope with unpredictable situations unless it relies completely on a tier infrastructure that can modify any of these models on the fly, then trigger the appropriate transformations to update the Final UI. This is the approach we have adopted for MARI [7]. In its current implementation, MARI provides a reasonable answer to Observations #1, #2, and #3: (a) the "fast-food" UI portions are generated from a task model. The corresponding AUI and CUI are automatically generated and maintained by MARI : they are not imposed on the developer; (b) In addition, hand-coded service-oriented components can be dynamically plugged and mapped to sub-tasks

whose UI cannot be generated by transformations. MARI has been applied to develop a photos browser that includes an augmented multi-touch table.

Alternatively, the various perspectives of the system (task models, AUI, FUI, context model, etc.) as well as the adaptation mechanisms may be distributed across distinct UI service-oriented components, each one covering a small task grain that can be run in different contexts of use. We have adopted this approach to implement the Comet toolkit [2]. Basically, a Comet is a plastic micro-interactive system whose architecture pushes forward the separation of concerns advocated by PAC and MVC. The functional coverage of a comet is left open (from a plastic widget such as a control panel, to a complete system such as a powerpoint-like slide viewer). Each Comet embeds its own task model, its own adaptation algorithm, as well as multiple CUI's and FUI's, each one adapted to a particular context of use. FUI's are hand-coded possibly using different toolkits to satisfy our requirements for fine-grained personalization and heterogeneity (Observation #3). From the infrastructure point of view, a Comet is a service that can be discovered, deployed and integrated dynamically into the configuration that constitutes an interactive environment.

Conclusion

The community has a good understanding about the nature of the meta-models for describing the high-level aspects of plastic interactive systems (e.g., task and domain-dependent concepts). Blurring the distinction between the design and the runtime phases provides humans (e.g., designers, installers, end-users) with full potential for flexibility and control. On the other hand, we still fall short at describing fine-grained post-WIMP

multimodal interaction and at supporting situations that could not be predicted at design time. For these cases, we claim that hand-coding and a tier service-oriented infrastructure are unavoidable. This is what we have done with MARI and the Comets, two very different ways of applying our principles for UI plasticity.

Acknowledgements

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. *In Proc. EUSAI 2004*, LNCS Vol. 3295, Springer-Verlag (Publ.) (2004), 291-302.
- [2] Calvary, G. Coutaz, J. Dâassi, O., Balme, L. Demeure, A. Towards a new generation of widgets for supporting software plasticity: the "comet". *In Proc. of Joint EHCI-DSV-IS (2004)*.
- [3] Calvary, G., Coutaz, J., Thevenin, D. A Unifying Reference Framework for the Development of Plastic User Interfaces. *In Proc. Engineering HCI'01*, Springer Publ., LNCS 2254 (2001), 173-192
- [4] Coutaz, J. Meta-User Interface for Ambient Spaces. *In Proc. TAMODIA'06*, Springer LNCS (2006), 1-15.
- [5] Kurtev, I., Bézivin, J. & Aksit, M. Technological Spaces: an Initial Appraisal. *In Proc. CoopIS, DOA'2002 Federated Conferences*, Industrial track, Irvine (2002).
- [6] Myers, B., Hudson, S.E. & Pausch, R.: Past, Present, and Future of User Interface Software Tools. *TOCHI*, ACM Publ., Vol 7(1), (2000) 3-28
- [7] Sottet, J.S., Calvary, G., Coutaz, J., Favre, J.M. A Model-Driven Engineering Approach for the Usability of Plastic User Interfaces. *In Proc. Engineering Interactive Systems 2007*, DSV-IS 2007), Springer.