Chapter 13

THE COMETS INSPECTOR

Manipulating Multiple User Interface Representations Simultaneously

Alexandre Demeure, Gaëlle Calvary, Joëlle Coutaz, and Jean Vanderdonckt
CLIPS-IMAG, BP 53, F-38041 Grenoble Cedex 9 (France)
E-mail: {Alexandre.Demeure, Gaelle.Calvary, Joelle.Coutaz, jeanvdd}@imag.fr
URL: http://www-clips.imag.fr/iihm
Tel.: +33 4 76 51 48 54 – Fax: +33 4 76 44 66 75

Abstract Three types of representation are typically produced during the User Interface (UI) development life cycle: a conceptual representation holding the models used for elaborating a UI, an internal representation concerning the code of the UI, and an external representation expressing the look and feel of the UI. While the end user typically manipulates the external representation only, the designer and the developer respectively focus on the conceptual and internal representations. The Comets Inspector gathers all three representations into a single environment, thus providing the user (actually, the designer and the developer; in the future, the end-user) with multiple views of a same UI simultaneously. Each representation is made observable and modifiable through one or many "mini-UIs". Consistency is ensured through mappings between these representations. From a methodological point of view, the benefit is the integration of three stakeholders' perspectives in a consensual and consistent way, enabling the exploration and manipulation of design alternatives at run time. In particular, when the context of use will be changing, the end-user will be able to inspect the UI capabilities and control its adaptation, thus sustaining explicit plasticity

Keywords: Abstract user interface model, Comet, Conceptual representation, External representation, Internal representation, Model-based design, Plasticity of user interfaces

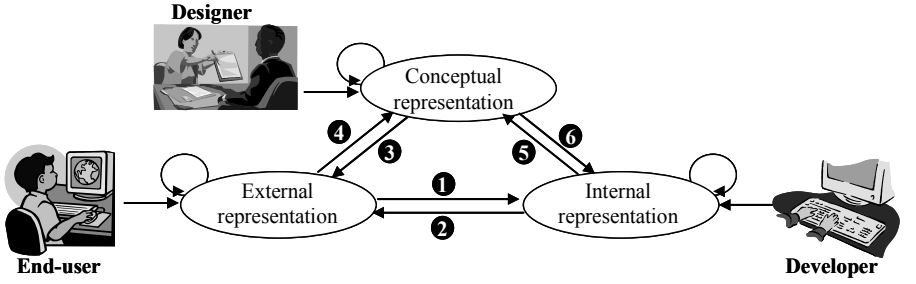
1. INTRODUCTION

Ubiquitous computing has led to new requirements in Human-Computer Interaction (HCI), in particular the need for interactive systems to adapt or

G. Calvary et al. (eds.), Computer-Aided Design of User Interfaces V, 167–174.

be adapted to their contexts of use while preserving usability. This *plasticity* property [5] has been studied for many years, mostly focusing on design time. In reality, neither the context of use nor the adaptation can always be envisioned at design time. As a result, part of plasticity has to be computed at run time. This paper presents an early prototype called the Comets Inspector that overcomes the plasticity issue. It favours the exploration of design alternatives both at design time and run time by embedding the all development life cycle of a User Interface (UI) in a single tool. It integrates three representations that traditionally need difficult conciliations between their stakeholders: the designer, developer and end-user. Fig. 1 elicits the three representations (inspired from [3,13]):

1. The *Conceptual Representation* models a UI using a given syntax and semantics according to consistent stylistics, which can be textual, graphical or both. This representation includes multiple models depending on the design method: typically task, domain, user, platform, environment, abstract and concrete UIs, etc. This representation is intended for the designer to capture UI requirements and information that will be turned into design options later on.
2. The *Internal Representation* consists of pieces of code programmed in an appropriate language (e.g., C, Tcl/Tk, Flash) for implementing a particular UI. This representation is typically the developer's one, where the UI code should reflect the design options decided by the designer.
3. The *External Representation* refers to the UI rendering which is visible and manipulable by the end-user. This rendering could be achieved through interpreters and/or code generators. This representation is the common view that is made visible to the end-user. The other representations are not usually.



❶ : Generation of the UI code from its appearance (see UIMSs, toolkits, interface builders).
❷ : Elaboration of the rendering from parts of the application/UI code.
❸/❻ : Generation of the UI appearance/code from the application's syntax and semantics (see MB-IDE).
❹/❺ : Recovering of design time models from the UI appearance/code (see reverse engineering).

Figure 1. The three representations of a UI making explicit six possible development paths

Fig. 1 makes explicit six possible development paths (1 to 6). Actually, these paths mainly suffer from five major shortcomings: 1) only one representation is available at a time; 2) most representations are limited to design time; 3) the mappings between all representations are not always ensured; 4) the manipulation of each representation is somewhat tedious; and 5) this manipulation is usually built-in. These five shortcomings are discussed in Section 2 with regard to the state of the art. They are turned into requirements that motivate both the Comet concept and the Comets Inspector described in Section 3. Section 4 discusses perspectives to the work.

2. RELATED WORK

FormsVBT [1] is probably the first manifestation of an environment combining more than one of the aforementioned representations at least: at design-time, the designer is able to manipulate T_eX specifications describing the look and feel of a graphical dialog box. This conceptual representation is directly mapped onto an external representation, a genuine UI which can be tested by the user for acceptance. The system also works the other way: when the external representation is affected, the conceptual representation is updated accordingly, thus maintaining a bijective mapping.

Teallach [2] supports the more general problem of maintaining mappings between representations [8] in a more sophisticated way: a conceptual representation is established based on a task and a domain models from which an external representation could be produced. The task or the domain models could be used separately or together. If an external representation is built manually, it is then possible to link it to the task and domain afterwards.

In [6], a methodology is developed that systematically produces a UI through a sequence of steps: task modelling, derivation of a dialog graph from the task model, and production of a final UI. Again, an external and an internal representation could be produced from a conceptual representation. The system does not work the other way: if the final UI is modified, these modifications are lost for the next re-generation. This problem is often referred to as the *round-trip engineering*. The situation is similar in [9].

An interesting idea introduced in [10] is to simultaneously provide the user with both the conceptual and external representations to the user so as to establish a more direct correspondence between the two views. This combination of views is maintained even at run-time. In [12], a forward engineering method is adopted to derive the internal and external representations from a conceptual representation that progressively goes from the task model to abstract presentations.

By examining these representative cases of the related work and other similar cases, we define five requirements to overcome the shortcomings elicited in Section 2:

- **Requirement n°1: all representations should be available simultaneously**, as opposed to one representation at a time or moving from one representation to another one as in FormsVBT [1] and Teallach [2].
- **Requirement n°2: all representations should be manipulated at run-time**, as opposed to design-time only. This is very relevant for propagating changes at run-time such as adaptations in case of plastic UIs. For instance, a UsiXML (http://www.usixml.org) specification could be conveyed and interpreted at run-time, including its adaptation rules that are embedded in the conceptual representation. But this is so far not supported in tools such as InterpiXML, the Java interpreter for UsiXML.
- **Requirement n°3: all representations should be coordinated in a consistent way**, as opposed to ensuring partial mapping or no mapping at all. Therefore in theory, six sets of mappings should be maintained (see Section 1). In practice, the requirement could be alleviated to be compliant with requirement number 2 that calls for an acceptable latency.
- **Requirement n°4: each representation should be manipulable via a dedicated "mini-UI"**, as opposed to other related works where the operations attached to each representation were not always salient. Various interaction styles could be relevant to take into account the different syntactic and semantic skills [7] of the stakeholders.
- **Requirement n°5: each "mini-UI" should be autonomous**, as opposed to tied up with the rest of the application and the environment.

The following section shows how the Comets inspector fully or partially addresses the five above requirements.

3. THE COMETS INSPECTOR

The notion of *Comet* has been fashioned from a software engineering perspective as *plastic interactors*, i.e. interactors capable of adapting or being adapted to the context of use while preserving usability [4,5]. A comet is "a self descriptive interactor that publishes the quality in use it guarantees for a set of contexts of use. It is able to either self-adapt to the current context of use, or be adapted by a tier-component. It can be dynamically discarded, respectively recruited, when it is unable, respectively able, to cover the current context of use" [5]. As opposed to Abstract Interaction Objects [14] and plastic widgets [11], a Comet is more powerful in that it embeds the alternate presentations depending on the context of use, the mechanisms to switch between them, and an underlying software architecture for controlling its behavior. A Comet comes with three facets [4,5]: Presentation, Abstraction,

and Control). Presentation and Abstraction are logical facets in charge of selecting the physical presentation/abstraction appropriate in the current context of use (Fig. 2).

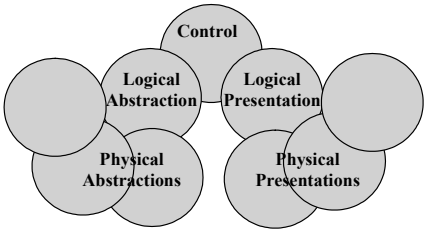


Figure 2. A comet, a software architecture construct made of a control, a logical abstraction and a logical presentation in charge of dealing with their potentially multiple physical abstractions and/or presentations. This "polymorphism" may be useful for adapting to the context of use

To address the five requirements introduced in Section 2, a Comets inspector has been designed and fully developed on top of the Tcl/Tk environment. Fig. 3 reproduces a screen shot of the inspector opened with a comet-based running example: the Home Heating Control System (HHCS).

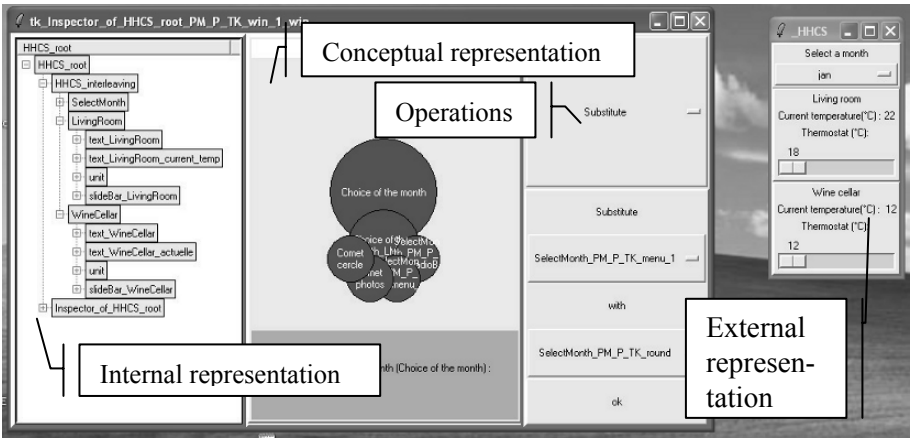


Figure 3. The three representations of a UI in the Comets Inspector. The selected comet in the conceptual representation is "Select a month"

HHCS (see the external representation on the right window in Fig. 3) is intended to help the user in managing the temperature at home. The user selects the month, browses the rooms and, if necessary, sets the thermostats of the rooms. The inspector makes observable the conceptual, internal, and external representations of HHCS. The conceptual representation (middle part of the left window in Fig. 3) depicts HHCS in terms of comets: both the control and the current/available logical and physical presentations are displayed (the abstractions have not been considered in this early version). The facets of the comets are depicted as circles according to Fig. 2. The internal

representation (left part of the left window in Fig. 3) materializes the hierarchy of Abstract Containers and Abstract Individual Components of the UI. The external representation consists of the direct rendering of the UI as perceived by the end-user. Thanks to a set of operations (right part of the left window in Fig. 3), the user can customize the UI. Let us suppose that when browsing the comet "Select a month" on the conceptual representation (see Fig. 3), the user perceives the existence of a round presentation. He/she simply selects both the "Substitute" operation and the round presentation. The external representation is immediately updated accordingly (Fig. 4). The Comets inspector addresses the requirements as follows:

- **Requirement n°1**: the three representations are available at any time as explained above.
- **Requirements n°2 & 3**: all three representations are manipulable at any time, whether it is at design- or at run-time. Consistency is ensured as illustrated on the "Substitute" operation (Figs. 3 and 4).

Figure 4. Choosing an alternate presentation (the round one for the "Select a month" comet) in the conceptual representation. The external representation is updated accordingly

Substitution can be performed between panels and windows. This is an easy way to implement detachable/(re)attachable UIs. For instance, in Fig. 5, a window-based presentation has been preferred for the comet "Set temperature of the living room". The UI has been detached accordingly in the external representation.

- **Requirements n°4 & 5**: As depicted in Fig. 3, each representation has its graphical autonomous mini-UI. However, they need further work to be usable by an end-user. It is also possible to manipulate each representation with Tcl/Tk commands, for instance for adding a comet to the hierarchy.

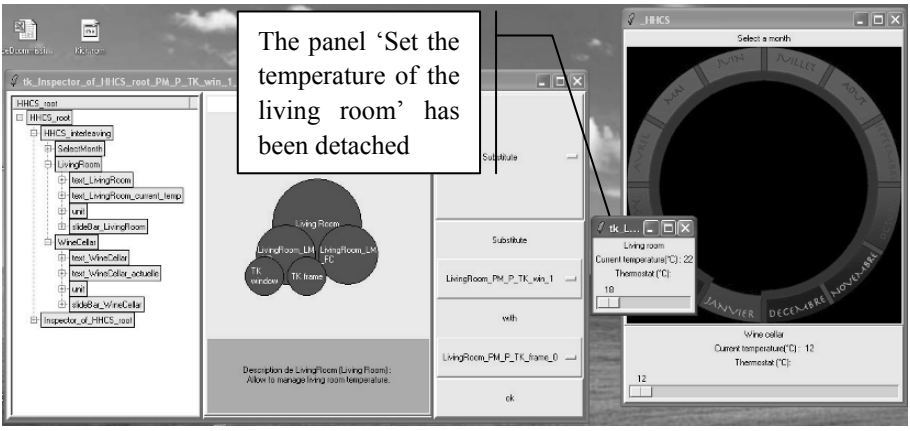


Figure 5. The Comets Inspector supports detachable UIs

4. CONCLUSION

Until now, the effort has been set on software architecture for both integrating the three stakeholders' perspectives, and supporting the polymorphism of comets. In the near future, the effort will be set on the conceptual representation so that comets could tell their tasks, concepts, structures and requirements in terms of context of use. After that, the focus will be set on UI in order to surpass the rapid prototyping tool and provide the end-user with a powerful tool for customizing his/her UI. Then, some evaluation will be conducted before providing a library of comets.

ACKNOWLEDGEMENTS

We gratefully acknowledge the support of the SIMILAR network of excellence (http://www.similar.cc), the European research task force creating human-machine interfaces SIMILAR to human-human communication (FP6-2002-IST1-507609). Jean Vanderdonckt would like to thank University Joseph Fourier which supported his position as invited professor for two months from May 2006.

REFERENCES

[1] Avrahami, G., Brooks, K.P., and Brown, M.H., *A Two-view Approach to Constructing User Interfaces*, in Proc. of the 16th Annual Conference on Computer graphics

and interactive techniques SIGGRAPH'89 (Boston, 31 July-4 August 1989), Computer Graphics, Vol. 23, No. 3, July 1989, pp. 137–146.

[2] Barclay, P.J., Griffiths, T., McKirdy, J., Paton, N.W., Cooper, R., and Kennedy, J., *The Teallach Tool: Using Models for Flexible User Interface Design*, in A. Puerta, J. Vanderdonckt (eds.), Proc. of 3rd Int. Conf. on CADUI'99 (Louvain-la-Neuve, 21–23 October 1999), Kluwer Academics Pub., Dordrecht, 1999, pp. 139–157.

[3] Barthet, M.-F., *The DIANE Method and its Connection with MERISE Method*, in Proc. of World Confererence "Ergonomic design, interfaces, products, Information" IEA'95 (Rio de Janeiro, 16–20 October 1995), pp. 106–110.

[4] Calvary, G., Dâassi, O., Coutaz, J., and Demeure, A., *Des Widgets aux Comets pour la Plasticité des Systèmes Interactifs*, Revue d'Interaction Homme-Machine, Europia, Vol. 6, No. 1, 2005, pp. 33–53.

[5] Calvary, G., Coutaz, J., Dâassi, O., Balme, L., and Demeure, A., *Towards a new Generation of Widgets for Supporting Software Plasticity: the "Comet"*, Proc. of 9th IFIP Working Conference on EHCI jointly with 11th Int. DSVIS Workshop, EHCI-DSVIS'2004 (Hamburg, July 11–13, 2004). Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 306–324.

[6] Dittmar, A., and Forbrig, P., *Methodological and Tool Support for a Task-Oriented Development of Interactive Systems*, in A. Puerta, J. Vanderdonckt (eds.), Proc. of 3rd Int. Conf. on Computer-Aided Design of User Interfaces CADUI'99 (Louvain-la-Neuve, 21–23 October 1999), Kluwer Academics Pub., Dordrecht, 1999, pp. 271–274.

[7] Jarke, M., and Vassiliou, Y., *A Framework for Choosing a Database Query Language*, ACM Computing Surveys, Vol. 17, No. 3, September 1985, pp. 313–370.

[8] Limbourg, Q., Vanderdonckt, J., and Souchon, N., *The Task-Dialog and Task-Presentation Mapping Problem: Some Preliminary Results*, in F. Paternò, Ph. Palanque (eds.), Proc. of 7th Int. Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'2000 (Limerick, 5–6 June 2000), Lecture Notes in Computer Science, Vol. 1946, Springer-Verlag, Berlin, 2000, pp. 227–246.

[9] Luyten, K., Clerckx, T., Coninx, K., and Vanderdonckt, J., *Derivation of a Dialog Model from a Task Model by Activity Chain Extraction*, in J. Jorge, N.J. Nunes, J. Falcão e Cunha (eds.), Proc. of 10th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2003 (Madeira, 4–6 June 2003), Lecture Notes in Computer Science, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 203–217.

[10] Navarre, D., Palanque, P., Paternò, F., Santoro, C., and Bastide, R., *A Tool Suite for Integrating Task and System Models through Scenarios*, in C. Johnson (ed.), Proc. of 8th Int. Workshop on DSV-IS'2001 (Glasgow, 13–15 June 2001), Lecture Notes in Computer Science, Vol. 2220, Springer-Vrlag, Berlin, pp. 88–113.

[11] Nylander, S., Bylund, M., and Waern, A., *The Ubiquitous Interactor – Device Independent Access to Mobile Services*, in R. Jacob, Q. Limbourg, J. Vanderdonckt (eds.), Proc. of 5th Int. Conf. of Computer-Aided Design of User Interfaces CADUI'2004 (Funchal, 13–16 January 2004), Kluwer Academics, Dordrecht, 2005, pp. 269–280.

[12] Paternò, F., and Santoro, C., *One Model, Many Interfaces*, in Ch. Kolski, J. Vanderdonckt (eds.), Proc. of 4th Int. Conf. on Computer-Aided Design of User Interfaces (Valenciennes, 15–17 May 2002), Kluwer Academics Publications, Dordrecht, 2002, pp. 143–154.

[13] Tarby, J.-C., *Gestion Automatique du Dialogue Homme-Machine à partir de Spécifications Conceptuelles*, Ph.D. thesis, Univ. of Toulouse I, Toulouse, 20 September 1993.

[14] Vanderdonckt, J., and Bodart, F., *Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection*, in Proc. of the ACM Conf. on Human Factors in Computing Systems INTERCHI'93 (Amsterdam, 24–29 April 1993), ACM Press, New York, 1993, pp. 424–429.