Made available by Hasselt University Library in https://documentserver.uhasselt.be

COMET(s), a software architecture style and an interactors toolkit for plastic User Interfaces

Non Peer-reviewed author version

DEMEURE, Alexandre; Calvary, G. & CONINX, Karin (2008) COMET(s), a software architecture style and an interactors toolkit for plastic User Interfaces. In: Graham, TCN & Palanque, P (Ed.) INTERACTIVE SYSTEMS: DESIGN, SPECIFICATION, AND VERIFICATION, PROCEEDINGS. p. 225-237..

DOI: 10.1007/978-3-540-70569-7_21

Handle: http://hdl.handle.net/1942/8511

COMET(s), A Software Architecture Style and an Interactors Toolkit for Plastic User Interfaces

Alexandre Demeure[1], Gaëlle Calvary[2], and Karin Coninx[1]

[1] Hasselt University - tUL - IBBT
Expertise Centre for Digital Media
Wetenschapspark 2, B-3590 Diepenbeek, Belgium
{alexandre.demeure, karin.coninx}@uhasselt.be
[2] Laboratoire LIG, 385, rue de la Bibliothèque - B.P. 53 –
38041 Grenoble Cedex 9, France
Gaelle.Calvary@imag.fr

Abstract. Plasticity of User Interfaces (UIs) refers to the ability of UIs to withstand variations of context of use (<User, Platform, Environment>) while preserving usability. This paper presents COMET, a software architecture style for building task-based plastic interactors. COMET bridges the gap between two main approaches in plasticity: model-driven engineering and interactors toolkits. Interactors that are compliant to the COMET style are called COMETs. These COMETs are multi-rendering multi-technological interactors (WIMP and post-WIMP, Web and non Web as well as vocal). COMETs are extensible and controllable by the user (up until now the designer, in the future the end-user). The COMET architecture and the use of COMETs are illustrated on an executable prototype: a slide viewer called CamNote++.

Keywords: Adaptation, context of use, plasticity, design alternatives, exploration, style sheets, tailored UIs, interactors.

1 Introduction

In the vision of ubiquitous computing users live in dynamic environments that change over time. Interactional, computational as well as communicational resources may arrive and disappear opportunistically. As these changes cannot always be foreseen at design time, there is a need for User Interfaces (UIs) to dynamically adapt to the actual context of use (<User, Platform, Environment>) while preserving usability. We use the term *Plasticity* [17] to denote this UI property. In this paper, we provide the designer (in the future the end-user) with tools for building plastic UIs and for exploring alternative renderings at design time as well as at runtime. The corner stone is a software architecture style called COMET (COntext Mouldable widgET) [3]. COMET compliant interactors are called COMETs.

COMETs are task-based interactors. They group together presentations that support a particular user's task. For instance, a set of radio buttons, a combo-box, a list and a pie menu (Fig. 1-A-1) support the user to "*select one option among N*". As a result, they are gathered in one and the same COMET which purpose is to select one

© Springer-Verlag Berlin Heidelberg 2008

option among N. In the same way, COMETs based on task operators are defined. For instance, the interleaving COMET groups together several presentations for rendering interleaving. This can be done by putting the interleaved subtasks side by side in a certain window (Fig. 1-A-1), by using multiple windows (Fig. 1-A-2) or a navigation interactor such as a menu (Fig. 1-B). These two kinds of COMETs rely on the same architectural style: COMET.

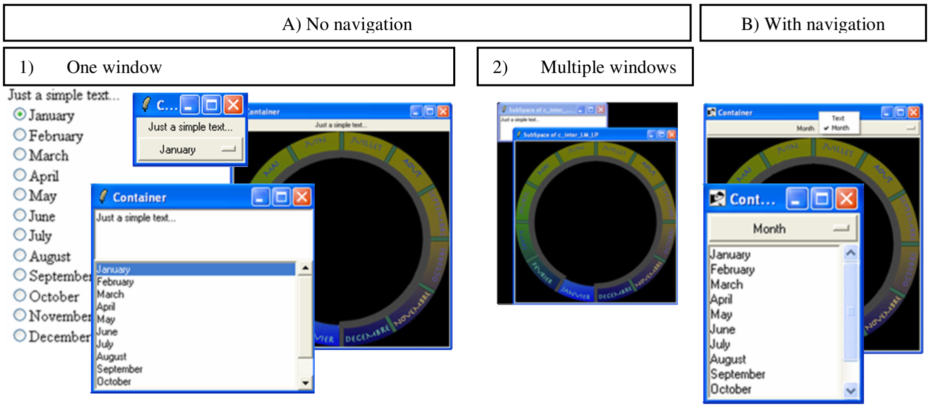

Fig. 1. Functionally equivalent interactors that vary from different points of view: navigation (A versus B), number of windows (A1 and B versus A2) and interactors presentations

COMET, the proposed architectural style, is fashioned for supporting polymorphism (i.e. multiple presentations) where presentations can belong to different rendering technological spaces (e.g. HTML, OpenGL, vocal). The goal of COMET(s) is to sustain the following four requirements:

- Sustaining UI adaptation at any level of abstraction: tasks and concepts, abstract, concrete and final UI as elicited in model-based approaches [2].
- The ability of UIs to be simultaneously rendered in several technologies including web and non web, WIMP and non WIMP, and also textual input and voice output.
- The ability of UIs to be dynamically transformed including enrichments with external and tailored UIs.
- The ability for the user (designer and/or end-user) to explore design alternatives by substituting presentations of COMETs at design time as well as at runtime.

Fig. 2 provides an overview of the global approach. The principles are threefold: (1) a UI is fully defined as a graph of COMETs, (2) the graph can be tuned through transformations, (3) transformations can take benefit from a semantic network [8] to retrieve components (COMETs as well as presentations of COMETs) and update the graph of COMETs accordingly.

This paper focuses on the graph of COMETs. The transformations are not described because of space. The semantic network is described in [8]. Section 2 presents the related work. Section 3 describes an executable demonstrator implemented with COMETs. Section 4 is devoted to the COMET architectural style. Finally, section 5 is about development using COMETs.

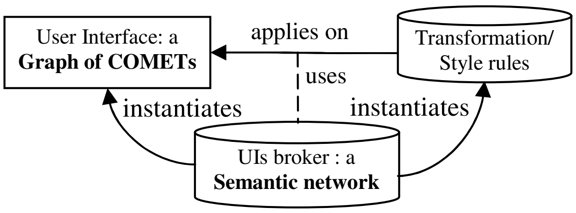

Fig. 2. An overview of the COMET-based approach

2 Related Work

In plasticity, the state of the art can be roughly categorized into three main approaches: Model Driven Engineering (MDE), window managers and widget toolkits.

MDE is probably the area [2,7,9] that recently received the most attention. Separation of concerns is the core principle. A UI is described from different perspectives (task, concepts, abstract UI (AUI), concrete UI (CUI), final UI (FUI)), each of them giving rise to a specific model. In the same way, the functional core and the context of use can be described along a set of models. Models are linked together through mappings. Mappings convey the widgets rationale (the tasks they support) as well as the UI deployment among the set of available platforms [4]. So far, MDE has been widely explored for the forward generation of standardized UIs.

Façade [16] investigates another approach: the adaptation is performed at the windows manager level. Adaptation is fully driven by the end-user who can dynamically copy/paste/replace parts of the UI. Façade is limited to graphical UIs. It relies on the widgets toolkit for the introspection mechanisms and the set of available widgets. As in practice none of these toolkits reaches the task level, adaptation can not be performed at a high level of abstraction (task or dialog).

Table 1. Analysis of the state of the art with regard to our four requirements

	Levels of abstraction	**Technological coverage**	**Extensibility**	**Controllability**
MDE [2,7,9]	All	Multiple	Hard	Depends on the underlying infrastructure
Windows manager [16]	CUI/FUI	Graphics	Irrelevant	End-user
ACE [11]	~Task	C++ toolkit	Easy	Designer
WAHID [10]	CUI	MFC	Hard	System
XFORMS	~Task/AUI	Web	Impossible	System with the help of the designer
FRUIT [12]	~Task	Depend on shells	Impossible	System
Multimodal Widgets [6]	~Task	Java/SWING	?	System with the help of the designer
Ubiquitous interactor [14]	~Task	Depends on interpreters	Impossible	System and designer

Widget toolkits have already extensively been explored. They tackle specific plasticity issues such as multimodality [6,12], polymorphism [10,11,14], or post-WIMP UIs [13]. None of these covers the tasks operators: sequence, interleaving, or operator, and so on [15]. As a result, all the transformations changing the way the navigation is rendered are lost (Fig.1) (e.g., switching from a menu to hyperlinks, tabbed panes, blanks or separators). In addition, only some approaches [11] support extensibility easily. Presentations are mostly mono-technological, and adaptation is neither foreseeable nor controllable.

Table 1 summarizes the state of the art with regard to the four abovementioned requirements. It shows that mixing MDE and widget toolkits may be promising for meeting all the requirements. This is the core principle of COMET(s).

3 CamNote++, A Running Demonstrator of COMET(s)

CamNote++ is a presentation software (like PowerPoint) that can be used by two kinds of users: speakers and spectators. CamNote++ is capable of adaptating to the screen size and takes into account hardware capabilities such as graphical hardware acceleration. Therefore, CamNote++ can be considered plastic with regard to the platform dimension of the context of use. CamNote++ is built with COMETs implemented in TCL. It can be rendered using several technologies according to the user's platform. For instance, if the user accesses CamNote++ via a web browser then AJAX/HTML is used. Both WIMP (e.g. form-based UIs) and/or post-WIMP UIs (e.g. multiple interaction points and speech UIs) can be used to render the application and interact with it. WIMP UIs rely on standard widgets available on the platform whereas post-WIMP UIs make use of toolkits such as OpenGL or Microsoft SAPI when available. WIMP and post-WIMP renderings can be used simultaneously.

From the end-user's perspective, CamNote++ first requires the user to log in (*Identify* task). The following tasks depend on the user's role: either speaker or spectator. In both cases, the current slide is rendered to the user. Two modes are available: presentation mode and question mode. The question mode corresponds to the case where the speaker is interrupted by someone for asking a question. In the presentation mode, only the speaker can control the viewer. In the question mode, spectators can also browse the slides using a dedicated controller. This is useful for supporting questions such as "*In slide N, what do you mean by …?*".

Fig. 3-1 shows CamNote++ in action for a speaker using a PC. The rendering is post-WIMP. At the beginning (A), CamNote++ is not operating in full screen mode: two windows are displayed to show both the current slide and the slides controller. When the speaker activates the full screen mode, the slide controller smoothly merges with the current slide (B) until being completely embedded in the slide (C). A picture of a keyboard is faded in and out (C) to make the user aware that he/she can now control the slide viewer using the physical keyboard (D). The keyboard controller is retrieved in the semantic network (a description of this approach is beyond the purpose of this paper).

Fig. 3-2 shows the web version of CamNote++ for a remote watcher. The current slide is updated using AJAX. In A, no style sheet is applied: the slide controller (in the upper part of the window) is composed of buttons and a dropdown menu for setting the current slide number. The current slide is displayed just beneath. An input field is placed at the bottom of the window to support taking notes. In B, a style sheet (specified by the designer) is applied for both improving the grouping (black boxes are added to better delimit workspaces) and for expanding the text area. In C, a tailored presentation is preferred for the slide controller: its container is a moveable translucent window. In D, the user's tasks (controlling the slides, perceiving the current slide and taking notes) are not directly observable in this case: they are browsable through tabbed panes. Style sheets (i.e., transformations) are not described in detail in this paper.

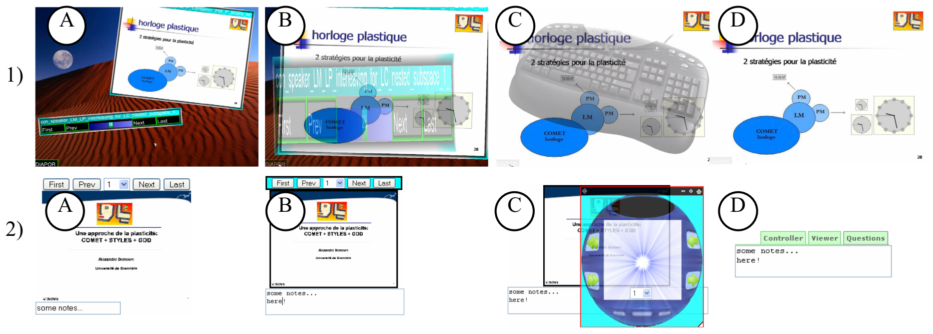
1)
2)
A
B
C
D
horloge plastique
2 stratégies pour la plasticité
COMET horloge
LM
PM
First
Prev
Next
Last
con_speaker_LM_LP_interleaving_for_LC_nested_subspace
Une approche de la plasticité: COMET + STYLES + GDD
some notes...
some notes... here!
Controller
Viewer
Questions

Fig. 3. 1) The OpenGL-based post-WIMP version of CamNote++ for the speaker. 2) The AJAX/HTML versions of CamNote++ for a spectator.

The next section describes the cornerstone of the toolkit: the COMET architectural style.

4 The COMET Style

COMET is driven by three principles: (1) Separation of concerns, (2) Reuse of existing toolkits (e.g., AJAX/HTML, TK, vocal, OpenGL), and (3) Recursivity so that a COMET can recursively be composed of COMETs.

This section describes the architectural style: first, the structure, then the event propagation. Finally we show how engineering interactive systems takes place when using COMET.

4.1 Structure

A COMET is composed of three facets. Each of them is responsible of one specific concern (Separation of concerns principle):

- A Logical Consistency (LC) represents the user's task (e.g., control the slides) or the task operator (e.g., interleaving) that the COMET supports. It denotes the semantics of the service that the COMET provides. The semantics gives rise to a specific API, called *semantic API* (e.g., next slide, previous slide...). The LC is associated to one or many Logical Models (LM). If many, LC is in charge of maintaining consistency between these LMs.

• A Logical Model (LM) is in charge of a specific concern related to the realization of the semantics. Usually, a distinction is made between the presentation and the abstraction (i.e., functional core). Whatever the concern is, each LM has to implement the semantic API of the corresponding LC (e.g., next slide...): this semantic API is the language that LC and LM share. The API can be extended to take into account specific concerns (e.g., blurring the slide). In turn, a LM is associated to one or many Physical Models (PM). If many, LM is in charge of maintaining consistency between these PMs. It also provides PM factories for instantiating PMs on the fly.

• A Physical Model (PM) is a specific means for realizing a LM. A presentation PM encapsulates the code of primitive toolkits such as OpenGL, HTML, SAPI, etc. (Reuse principle). A functional PM would encapsulate network protocols (e.g., AIM, MSN, YAHOO, IRC, etc.) in case of a Chat COMET. Encapsulated codes are called *technological primitives*. A PM has to implement its LM semantic API: this API is the shared language. A PM also describes the context of use it requires (e.g. JAVA, screen size, etc.).

LC, LM and PM are called *nodes*. Nodes can be tagged with decorations. For instance, a LC can be tagged as being frequent or critic according to the task decorations in the task model. A LM can be tagged with the concern it is in charge of (e.g., presentation). A PM can be tagged with the interaction path length it requires for achieving the task. Fig. 4 depicts the COMET architecture style as an UML class diagram (A) and in a dedicated graphical representation (B).

Constraints ensure that a node can only be plugged with compatible ones: LCs with LCs, LMs with LMs, PMs with technological compatible PMs (e.g. HTML presentations).

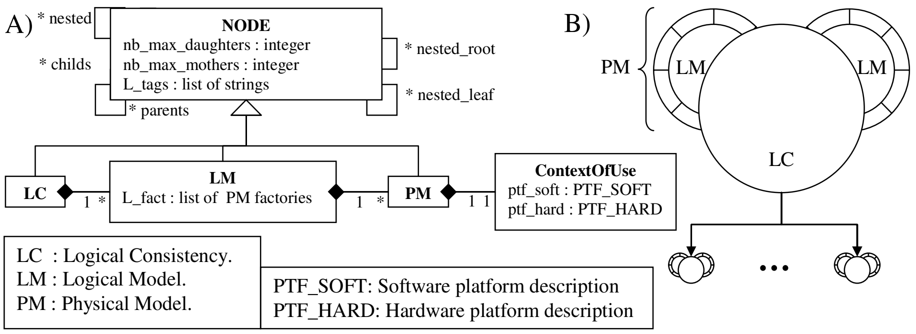

Fig. 4. The COMET architectural style: A) A UML class diagram. B) A dedicated graphical representation.

In the following we take the CamNote++ "*Remote Controller*" COMET as an example. Several presentations can be envisioned (Fig. 5-A) using different technologies: vocal, web, post WIMP, etc. Each presentation gives rise to a specific presentation PM. From a functional point of view, the controller can convey commands using different network protocols (Fig. 5-B).

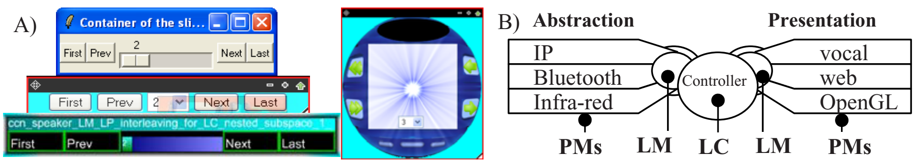

Fig. 5. A) Few presentations for the Remote Controller COMET. B) A graphical representation of the Remote Controller COMET.

Consistency among the different facets is ensured by a communication mechanism based on event propagation.

4.2 Events Propagation Inside a COMET

Events may be fired by two sources: either by a program that calls a COMET's function (e.g. set the current slide number) via its LC (Fig. 6-A) or by the user interacting with a PM (e.g. via the OpenGL presentation of the CamNote++ slides controller) thus triggering an event (Fig. 6-B). Each time an event is triggered, it is propagated along the COMET to the other facets in order to ensure consistency (Fig. 6). Ensuring consistency among presentation PMs can be seen as a multimodality issue if presentation PMs are seen as interaction modalities and multimodality as a combination of modalities.

The CARE properties [5] provide a framework for reasoning about the combination of modalities. Only Redundancy and Equivalence are addressed yet in COMET. Assignment is out of scope of our work presented in this paper. Complementarity as defined in the "put that there" paradigm [1] goes far beyond our work. Only basic forms of complementarity are covered up until now: (1) Input complementarity of modalities is used to achieve an elementary task. For instance, the task "Specify text" is achieved by alternatively using a keyboard-based and a voice-based PM. COMET supports this by design. (2) Input complementarity of modalities to achieve composed tasks (e.g. typing text and changing its colour). It is possible to use different modalities for the different sub-tasks. Again, COMET supports this by design. (3) Finally, output complementarity is achieved by using several PMs for a presentation LM.

To support Redundancy (R) and Equivalence (E), we have defined a domain specific language: COMET/RE (R for Redundancy and E for Equivalence). The idea is to associate a COMET/RE sentence to each function of the semantic API of a presentation LM. These sentences specify the way events must be processed. For instance, "R(E(gfx), E(vocal))" associated to the function F (e.g. switch to diaporama mode) means that the call of F has to be propagated to the LC if and only if one graphical PM (gfx) and one vocal PM (vocal) at least (E) are used in a redundant way (R). In case of redundancy, the propagation to the LC is conditioned by the activation of the corresponding PMs. In case of equivalence, the propagation to the LC is done as soon as an equivalent PM is activated. Fig. 6-B illustrates the COMET/RE sentence "E(*)": it means that all PMs (*) are equivalent (E) for F.

The next subsection elaborates on interactive systems as graphs of COMETs.

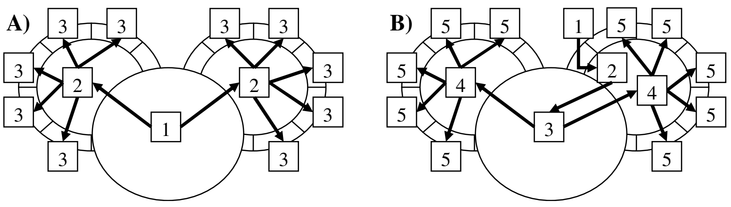

Fig. 6. Propagation of events (arrows) inside a COMET. Numbers represent the calls ordering. A) Propagation starting from the LC. B) Propagation starting from a PM. The propagation from 2 to 3 depends on the evaluation of the associated COMET/RE sentence.

4.3 Graphs of COMETs

Using COMETs, an interactive system is a graph of COMETs. More precisely, there are three types of interconnected graphs: a graph of LCs, a graph of presentation LMs and a set of graphs of presentation PMs, one per PM rendering technology (TK, OpenGL, etc.) as for instance a TK PM can only be rendered inside another TK PM. In all the graphs, the "parent-child" relation has the same meaning: the child expresses itself with regard to its parent (e.g. a PM child is rendered in the PM parent).

Consider CamNote++ for example. Fig. 7-A depicts the graph of COMETs for the spectator's UI: a text specifyer (to take notes), a slide controller and a slide viewer are interleaved. All the LCs are linked together in a graph. All the presentation LMs are linked together in another graph. All the presentation PMs are linked together in mono-technological graphs (one for TK, one for vocal, etc.). COMET ensures the interconnection between these graphs. For readability, only the graph of LCs is depicted in Fig. 7-A. Fig. 7-B shows the rendering of the AJAX/HTML-based graph of PMs.

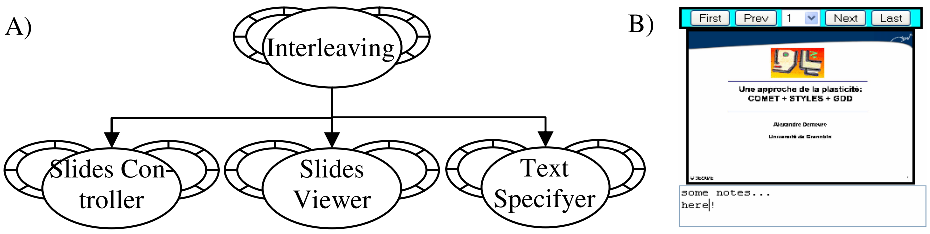

Fig. 7. A) Graph of COMETs for a spectator. B) A corresponding AJAX/HTML UI

Each node that contains a graph of COMETs (Recursivity principle) is said to be composite by opposition with atomic nodes (which do not contain a graph). Fig. 8 illustrates how recursivity is used in the CamNote++ COMET. The LC part of the COMET is composed of COMETs that correspond to the different roles (speaker or spectator) of CamNote++ users. All the COMETs (speaker or spectator) share a same COMET slides viewer, thus ensuring the slides synchronisation among users. Besides the recursivity in the LC, there is a recursivity of presentation PMs. Each PM of

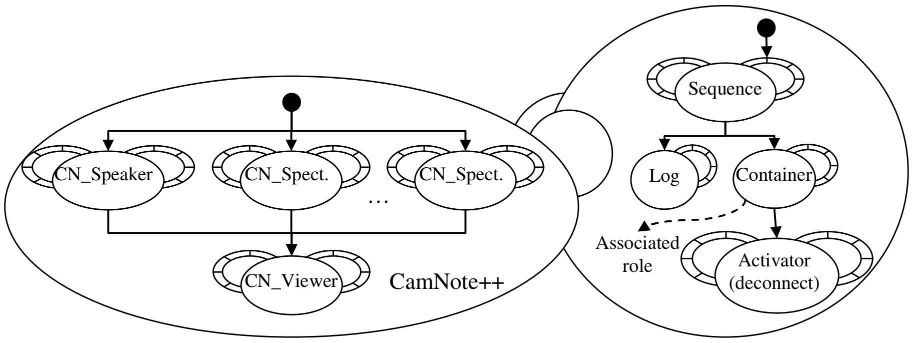

Fig. 8. The CamNote++ COMET. The composite PM is in charge of log in the user to the right role (speaker or spectator). The composite LC manages the different roles (modeled with dedicated COMETs). All the roles share a same COMET slides viewer.

CamNote++ is in charge of identifying the user and setting his/her role. In practice, each time a user accesses CamNote++ by mean of a new UI (e.g. when opening a web browser), he/she is asked to identify his/herself so that CamNote++ can display the right UI (speaker or spectator).

There is no straightforward rule to know when and how to use composite nodes. It is up to the designer to decide about using this feature. However, we can say that task decomposition is likely to be translated into a composite LC; workspaces organisation is likely to be translated into a composite presentation LM, and widgets decomposition is likely to be translated into a composite presentation PM. As shown in Fig. 8, a composite PM can also be used to manage access to a COMET for different users.

In practice, designers only have to specify the graph of LCs. The presentation LM graph (respectively PM graph) is automatically generated according to the LC graph (respectively LM graph). The graph of PMs is built with respect to the context of use: an AJAX/HTML PM is plugged into AJAX/HTML compatible PMs. Note that graphs of presentation LMs and PMs are automatically generated. Indeed, COMETs always contain presentation facets. This is not the case for other facets such as abstraction.

5 Developing with COMETs

This section puts the COMET style in action. Three kinds of requirements are considered to show how COMET can be used for tuning CamNote++ and target additional contexts of use.

5.1 Distributing the Slides Controller on a PDA

Imagine the designer decides to distribute CamNote++ (for the speaker role) on a PC and a PDA: the Slides Viewer on the PC using OpenGL; the Slides Controller on the PDA using HTML. To do this, the designer only needs to plug an OpenGL and an

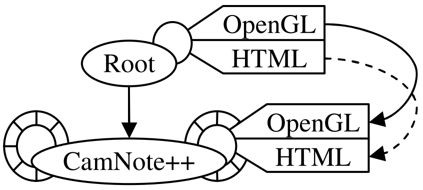

Fig. 9. Graph of COMETs corresponding to CamNote++ rendered in OpenGL and HTML. Links (arrows) between presentation PMs are automatically generated based on the LC links.

HTML PM to the COMET Root which expresses that the graph of COMETs will be rendered using these two technologies (Fig. 9).

Once the graph (Fig. 9) is built, the designer configures the presentations to be rendered. For instance, he/she specifies that the HTML Slides Controller has to fit the web page. This can be done using a style/transformation rule that, if necessary, calls the semantic network for retrieving presentations. Fig.10 provides an example without any detail about the syntax. The example (A) asks for replacing the HTML slides controller with a skinable version (B) to be retrieved in the semantic network.

A)

```
#CN_Speaker->PMs[soft_type == HTML](SlideController) {
  type          :          SlideController_CUI_skinnable;
 }
```

B)

Fig. 10. A) A transformation rule for substituting the HTML presentation of the Speaker's Slides Controller by the one shown in B

5.2 Requiring Redundancy for Switching the Presentation Mode

Imagine switching between full screen and window-based modes appears to be a critical task. Requiring redundancy for changing the mode may be an option to prevent the user from making errors,. In that case, the speaker has to ask for a switch using both the HTML and OpenGL UIs. Such a modification can simply be done using a single transformation rule (Fig. 11). This rule specifies that the mode activator COMET can only be activated if both the OpenGL and HTML presentations are activated in the same temporal window of 2000 milliseconds.

```
#CN_Speaker(Activator.DIAPORAMA->_LM_LP)  {
 COMET_RE_expr : activate  R(2000,E(HTML),E(OpenGL)) ;
}
```

Fig. 11. A transformation rule for requiring redundancy between OpenGL and HTML presentations when switching between full screen and window-based modes

5.3 Integrating the Pixels Mirror Feature into the OpenGL Slides Viewer

Imagine the designer decides to include a pixels mirror when possible (i.e., in case a camera is connected to the PC). Using COMET, this is achieved either at design time or at runtime by (1) encapsulating the OpenGL "Slides Viewer" presentation PM into a composite PM, adding a Video COMET in charge of displaying the camera images, and adding an integer Choice COMET to set the translucence level of the video (first rule in Fig. 12-A, *"Eval : U_encapsulator_PM $obj "Container(, \$core, Video(), ChoiceN(set_range \"0 100\"))";"*). Then (2) the COMET choice is linked to the video OpenGL presentation PM so that every time a new value is set, the translucence level is updated accordingly (second rule of Fig. 12-A, an Event Condition Action is defined by *"ECA : set_current, true, set video [CSS++ "#CN_Speaker->PMs[type==OpenGL] CN_Viewer(Video)"] --- $video set_translucidity [expr $value / 100.0];"*). Finally, the last two rules express how the presentations are laid out. Fig. 12-B graphically describes the COMET Slide Viewer before and after applying the rules.

A)

```
#CN_Speaker->PMs[type==OpenGL] CN_Viewer {
  Eval : U_encapsulator_PM  $obj "Container(, \$core, Video(), ChoiceN(set_range \"0 100\"))";
}
#CN_Speaker->PMs[type==OpenGL] CN_Viewer(ChoiceN) {
  ECA : set_current, true
        , set video [CSS++ "#CN_Speaker->PMs[type==OpenGL] CN_Viewer(Video)"] ---
          $video set_translucidity [expr $value / 100.0];
}
#CN_Speaker->PMs[type==OpenGL] CN_Viewer(Container, Video)  {
  Layout : Fit_parent;
}
#CN_Speaker->PMs[type==OpenGL] CN_Viewer(ChoiceN)  {
  Type : Slider;
  Layout : Bottom;
}
```

B)

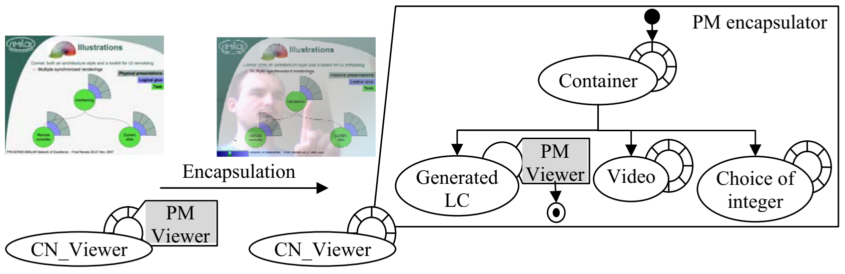

Fig. 12. Four transformation rules, a dozen of lines of code to integrate the pixels mirror feature in CamNote++

6 Conclusion and Future Work

In this paper, we present COMET, a new software architecture style specially crafted for plasticity. COMET bridges the gap between two main research areas in plasticity:

MDE and interactors toolkits. COMET meets four main requirements that had never been simultaneously satisfied so far. The four levels of abstraction and the multi rendering feature are ensured by design concepts: tasks-concepts, AUI, CUI and FUI are respectively embodied in LCs, presentation LMs, PMs and technological primitives. Technological primitives target different languages and toolkits in a non exclusive way. Extensibility and controllability are satisfied with two additional tools (not described in this paper): style sheets for specifying transformations, and a semantic network for retrieving existing UI elements.

The COMET style has been implemented in TCL giving rise to a COMETs toolkit that contains classical interactors (e.g., select one option among N) as well as more innovative ones in charge of task operators (e.g., interleaving, sequence). Each interactor can be polymorphic including exotic custom-made presentations. In turn, the COMETs toolkit has been used for implementing CamNote++, an executable plastic presentation software that illustrates the architecture and concepts proposed in this paper. We show the powerful COMET capabilities for extending and tuning UIs, and for exploring design alternatives. This can be done both at design time and at run time.

In the future, we aim at exploring UIs for visualizing and transforming COMETs at runtime. We keep in mind the difficult issue of evaluating the architecture model and the toolkit. Using the proposed approach in teaching situations could provide an initial evaluation.

Videos are available at **http://iihm.imag.fr/demeure/.**

Acknowledgments. Part of the research has been funded by the SIMILAR European network and ERDF (European Regional Development Fund), the Flemish Government and the Flemish Interdisciplinary institute for BroadBand Technology (IBBT).

References

1. Bolt, R.A.: "Put-That-There": Voice and Gesture at the Graphics Interface. Computer Graphics 14(3), 262–270 (1980)
2. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi-Target User Interfaces. Interacting With Computers 15/3, 289–308 (2003)
3. Calvary, G., Coutaz, J., Dâassi, O., Balme, L., Demeure, A.: Towards a New Generation of Widgets for Supporting Software Plasticity: The "Comet". In: Bastide, R., Palanque, P., Roth, J. (eds.) DSV-IS 2004 and EHCI 2004. LNCS, vol. 3425, pp. 306–324. Springer, Heidelberg (2005)
4. Clerckx, T., Luyten, K., Coninx, K.: The mapping problem back and forth: customizing dynamic models while preserving consistency. In: Proceedings of the 3rd Annual Conference on Task Models and Diagrams, TAMODIA 2004, November 15 - 16, 2004, vol. 86, pp. 33–42. ACM Press, New York (2004)
5. Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., Young, R.: Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE properties. In: Arnesen, S.A., Gilmore, D. (eds.) Proceedings of the INTERACT 1995 conference, June 1995, pp. 115–120. Chapman&Hall Publ., Lillehammer (1995)

6. Crease, M., Brewster, S.A., Gray, P.: Caring, sharing widgets: a toolkit of sensitive widgets. In: 14th Annual Conference of the British HCI Group, Sunderland, England, September 5-8, 2000. British Computer Society conference series, pp. 257–270 (2000)
7. da Silva, P.: User Interface Declarative Models and Development Environments: A Survey. In: Palanque, P., Paternó, F. (eds.) DSV-IS 2000. LNCS, vol. 1946, pp. 207–226. Springer, Heidelberg (2001)
8. Demeure, A., Calvary, G., Coutaz, J., Vanderdonckt, J.: The COMETs Inspector: Towards Run Time Plasticity Control Based on a Semantic Network. In: Coninx, K., Luyten, K., Schneider, K.A. (eds.) TAMODIA 2006. LNCS, vol. 4385. Springer, Heidelberg (2007)
9. Gajos, K., Weld, D.: Preference elicitation for interface optimization. In: UIST 2005: Proceedings of the 18th annual ACM symposium on User interface software and technology, Seattle, WA, USA, pp. 173–182 (2005)
10. Jabarin, B., Graham, N.: Architectures for Widget-Based Plasticity. In: Jorge, J.A., Jardim Nunes, N., Falcão e Cunha, J. (eds.) DSV-IS 2003. LNCS, vol. 2844, pp. 124–138. Springer, Heidelberg (2003)
11. Johnson, J.: Selectors: going beyond user-interface widgets. In: CHI 1992: Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 273–279 (1992)
12. Kawai, S., Aida, H., Saito, T.: Designing interface toolkit with dynamic selectable modality. In: Proceedings of the Second Annual ACM Conference on Assistive Technologies Assets 1996, April 11 - 12, 1996, pp. 72–79. ACM Press, New York (1996)
13. Lecolinet, E.: A molecular architecture for creating advanced GUIs. In: Proceedings of the 16th Annual ACM Symposium on User interface Software and Technology UIST 2003, November 02 - 05, 2003, pp. 135–144. ACM Press, New York (2003)
14. Nylander, S., Bylund, M., Waern, A.: The Ubiquitous Interactor – Device Independent Access to Mobile Services. In: Proc. of 5th Int. Conf. of Computer-Aided Design of User Interfaces CADUI 2004, January 13-16, 2004, pp. 269–280. Kluwer Academics, Dordrecht (2005)
15. Paterno', F., Mancini, C., Meniconi, S.,, C.: A Diagrammatic Notation for Specifying Task Models. In: Proceedings Interact 1997, Sydney, pp. 362–369. Chapman & Hall, Boca Raton (1997)
16. Stuerzlinger, W., Chapuis, O., Phillips, D., Roussel., N.: User Interface Façades: Towards Fully Adaptable User Interfaces. In: Proceedings of UIST 2006, October 2006, pp. 309–318. ACM Press, New York (2006)
17. Thevenin, D., Coutaz, J.: Plasticity of User Interfaces: Framework and Research Agenda. In: Edinburgh, A.S., Johnson, C. (eds.) Proc. Interact 1999, pp. 110–117. IFIP IOS Press Publ., Amsterdam (1999)