Plasticity of User Interfaces: A Revised Reference Framework

[1]*Gaëlle Calvary, Joëlle Coutaz, David Thevenin*
[2]*Quentin Limbourg, Nathalie Souchon, Laurent Bouillon, Murielle Florins, Jean Vanderdonckt*

[1]CLIPS-IMAG
BP 53
F-38041 Grenoble Cedex 9, France
Phone: +33 4 76 51 48 54
{gaelle.calvary,joelle.coutaz,david.thevenin}
@imag.fr

[2]Institut d'Administration et de Gestion - BCHI
Université catholique de Louvain
Place des Doyens, 1
B-1348 Louvain-la-Neuve, Belgium
{limbourg, souchon, bouillon,florins,
vanderdonckt}@isys.ucl.ac.be

ABSTRACT
Mobility coupled with the development of a wide variety of access devices has engendered new requirements for HCI such as the ability of user interfaces (UIs) to adapt to different contexts of use. We define a context of use as the set of values of variables that characterize the computational device(s) used for interacting with the system as well as the physical and social environment where the interaction takes place. A UI is plastic if it is able to adapt to context changes while preserving usability. In this paper we present a reference process for the engineering of plastic user interfaces. The process revises a previously published reference framework [3]. The amendment is twofold: first it refines the design time process and secondly extends its coverage to the run time.

Keywords
Human computer interaction, plasticity, adaptation, context of use, platform, environment, engineering, reference framework.

INTRODUCTION
Information technology is an increasingly essential part of the fabric and activity of our lives. Jini-enabled information appliances, XML approaches to information modeling [18] and rendering, and gateways between Internet and wireless network protocols [14] are being developed to cope with the pregnancy of the technical push. This exploratory development of novel devices and techniques is valuable in the short run. The approach, however, is not replicable and provides poor guidance to sustain future development of usable interactive technologies. As a result, there is a risk of a shortfall between technical promise and effective interaction. *Theories and principles developed so far in HCI must not be lost in the evolution!*

Although principles of user-centered design methods and modeling techniques [19] offer a sound substrate, pervasive computing opens the way to new challenging requirements. In particular, *people want to have the choice*. They want to be able to choose among a wide range of software platforms and hardware devices to accommodate multiple needs depending on places and spaces across time. Providing different interfaces specially crafted for each type of device and modality combination is extremely costly and could result in users having many different versions of interfaces on different devices. The impact includes massive under-use of interfaces potential and excessive development costs to maintain versions consistent across multiple platforms. The notion of *plasticity* to cope with these problems is introduced [23,24].

PLASTICITY
The term "plasticity" is inspired from the property of materials that expand and contract under natural constraints without breaking, thus preserving continuous usage. Applied to HCI, plasticity is the capacity of an interactive system to withstand *variations of context of use* while *preserving usability*. A *context of use for a plastic system* covers two classes of attributes:

- The attributes of the physical and software platform(s) used for interacting with the system. Typically, screen size and network bandwidth have an impact on the amount and modality of information to be rendered and transferred;
- The environmental attributes that describe the physical surroundings of the interaction. These include the set of objects, persons and events that are peripheral to the current task(s) but that may have an impact on the system and/or the user's behavior, either now or in the future. Typically, light conditions may influence the robustness of a computer vision-based tracking system, noisy environments may eliminate sonic feedback, etc. At the task level, location in space provides context for information relevance; tasks that are central in the office (e.g., writing a paper) may become secondary in a train, etc.

A *plastic user interface preserves usability* if the properties selected at design time to measure its usability are kept within a range of values as adaptation occurs to contextual changes. Although the properties developed so far in HCI [11] provide a sound basis, they do not cover all

aspects of plasticity. For example, they do not express the need for continuity [10] when migration occurs between contexts of use. Thus, we *need to extend and refine our body of properties* to cope with the new situations offered by the technology.

Activity theory takes into account the situation of action early in the design process. Unfortunately, situation-dependent information is lost in the development process due to the lack of appropriate notations of the design and development tools. As a result, current tools implicitly assume that users are working with a desktop computer located at a specific place. A notable exception is the context toolkit [21] developed for encapsulating sensors at the right level of abstraction and the "literate development" [5]. Although within the scope of plasticity, context toolkits cover low-level technical concerns only, and the literate development is not precise enough to address our problem. Therefore *a process model that supports the development of plastic user interfaces is required*. To support the description of the framework, we first present a case study provided by EDF, the French Electricity Company. We then remind the initial version of the framework and motivate its revision. It gives rise to a new version that is described in the last part of the paper.

AN EXAMPLE: A HOME HEATING CONTROL SYSTEM

The heating control system envisioned by EDF (The French Electricity Company) will be controlled by users situated in diverse contexts of use. These include:

- At home, through a dedicated wall-mounted device or through a Palm-like device connected to a wireless home-net.
- In the office, through any Web browser on a PC.
- Anywhere using a WAP-enabled mobile phone.

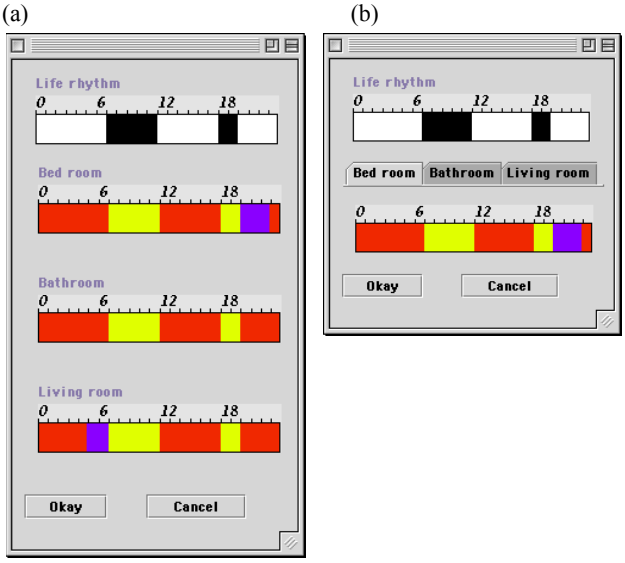

Figure 1. Large screen. Temperature of the rooms are available at a glance. (a) – Small screen. Temperature of one room is displayed at a time (b).

A typical user's task consists of consulting and modifying the temperature of a particular room. Fig. 1 and 2 show versions of the same system for the PC browser and the WAP-enabled phone respectively.

- In Fig. 1a, the system displays the current temperature for each room of the house. The screen size is comfortable enough to make *observable* the entire system state.
- In Fig. 1b, the system shows the temperature of a single room at a time. A thumbnail allows users to switch between rooms. In contrast with Fig. 1a, the system state is *browsable* due to limited screen size. As a result, additional navigational tasks are required to give access to the desired information.

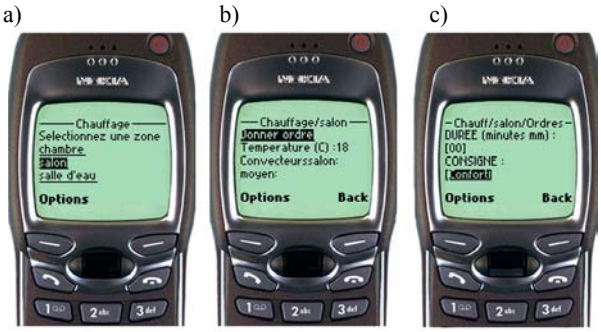

Figure 2. Modifying the temperature using a WAP-enabled mobile phone.

Fig. 2 shows the interaction pathway for setting the temperature of a room with a WAP mobile phone. The user selects the room, (e.g., "le salon" - the living room in Fig. 2a) and the system shows its current temperature (Fig. 2b). By selecting the editing function ("donner ordre"), one can modify the temperature (Fig. 2c). When comparing with the situation depicted in Fig. 1, navigation tasks have been introduced and a title for every deck (i.e., WML page) has been added to recall the user with the current location within the interaction space. All of these alternatives have been produced using the initial reference framework [3,4].

THE INITIAL REFERENCE FRAMEWORK

Generally speaking the reference framework is intended to serve as a reference instrument to help designers and developers to structure the development process of context-sensitive interactive systems, including plastic UIs. To this end, we adopted a model-based approach [19,20]:

- We built upon models used in current practice: concepts, tasks, task oriented specification (called Concepts and Tasks Model in Fig. 3 and 13), abstract and concrete user interfaces.
- We improved existing models to accommodate variations of the context of use: concepts and task oriented specification.
- We explicitly introduced new models and heuristics that have been overlooked or ignored so far to convey the context of use: for example, the platform and environment models.

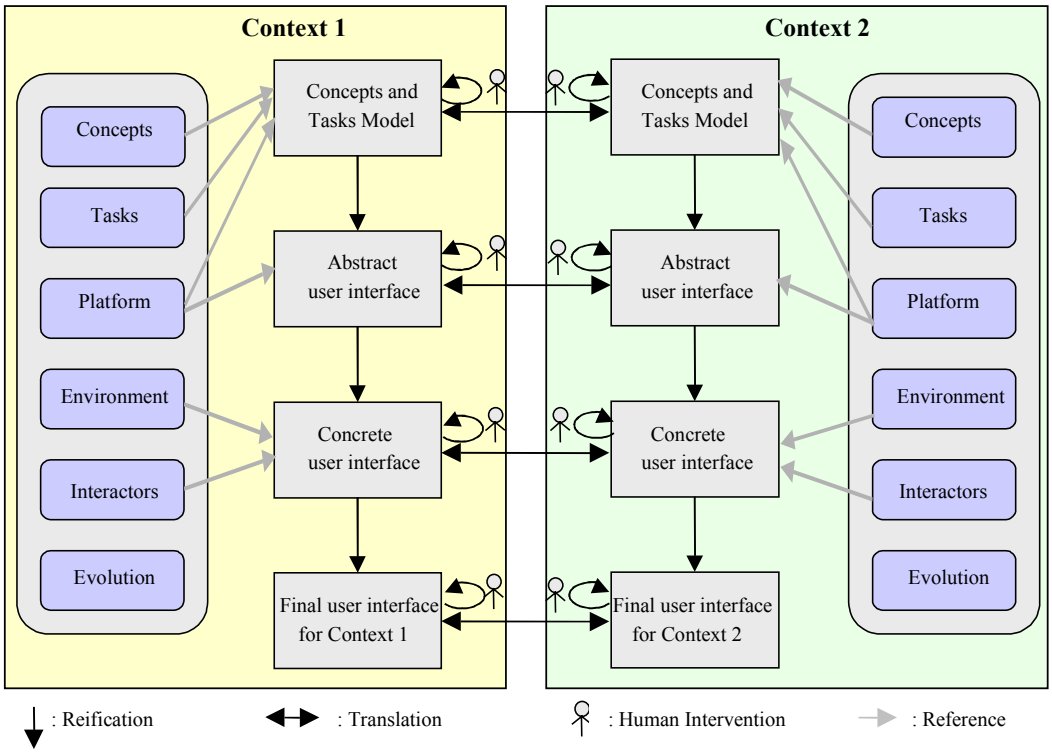

Figure 3. The initial reference development process for supporting plastic interactive systems.

Fig. 3 shows which models and how they are involved in the process. The *Concepts Model* captures information related to the domain of discourse: in this sense, it is similar to domain modeling. For example, for the Home Heating Control System it includes the concepts of life rhythm, room and temperature. The *Task Model* describes how a user is carrying out a task to fulfill the goals of the task. For example, programming the comfort at home consists in specifying both the user life rhythm and the temperature of the living room, bedroom and bathroom.

The *Platform Model* and the *Environment Model* define the contexts of use intended by the designers. For example, the size of the screen would be described in a platform model, whereas the level of noise of a room would be captured in an environment model. The *Interactors Model* describes "resource sensitive multimodal widgets" [1] available for producing the concrete interface. For example the labels, time-bars and buttons that appear on Fig. 1. The *Evolution Model* specifies the change of state within a context of use as well as the conditions for entering and leaving a particular context. For example, switching on the fly from the PDA to the wall-mounted device when the battery of the PDA gets low.

Each of the above models is referenced along the development process from the task specification to the running interactive system. The process is a combination of vertical reification and horizontal translation. *Vertical reification* covers the derivation process, from top level abstract models to run time implementation. *Horizontal derivations*, such as those performed between HTML and WML content descriptions, correspond to translations between models at the same level of reification. Reification and translation may be performed automatically from specifications, or manually by human expert designers, depending on the tools available, or by a combination of both with a mixed-initiative locus of control.

The richness of the reference framework relies in its capability to be instantiated in many ways:

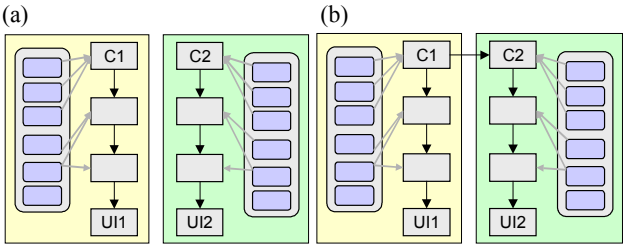

Figure 4. Independent reifications (a) ; Initial translation before reification (b).

- Fig. 4a represents the *most currently found practice*: two running systems are reified in parallel using separate input models, each one being specified for separate contexts of use (C1 and C2 in Fig. 4a). Unfortunately, there is not necessarily a way to factoring out common elements across both contexts of use, thus requiring maintaining some consistency between the multiple versions at the same time. For example, this approach is used when three different UI versions for the same task are developed simultaneously (e.g., in HTML, WML, and XML), but independently of each other.
- Fig. 4b depicts the *highest level approach* graphically: the task-oriented specification valid for one context C1

is translated to fit another context C2 at the task and concepts level. From there, reifications are performed in parallel. This approach has been followed for the Home Heating Control System using a WAP mobile phone (Fig. 2). Sub trees that correspond to infrequent tasks have been pruned from the original task tree developed for the Java-enabled platform. Because ARTStudio does not yet support Web-based techniques, the reification steps have been performed manually by a human expert. A similar approach is described in [6,7]. Another example of translation performed at the highest level occurs in [8]: domain concepts and task elements are defined in an ontology editor of the domain of discourse. These concepts, along with their attributes and relationships, are attached with production rules stating properties depending on the context of use (e.g., type of user, interaction style). Such rules include translations from one context C1 to another context C2 stating how tasks and their related concepts change when the context changes (Fig. 5).

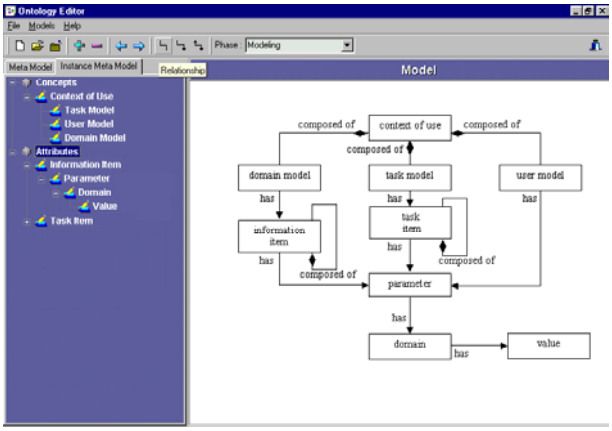

Figure 5. Ontology editor for task and concepts.

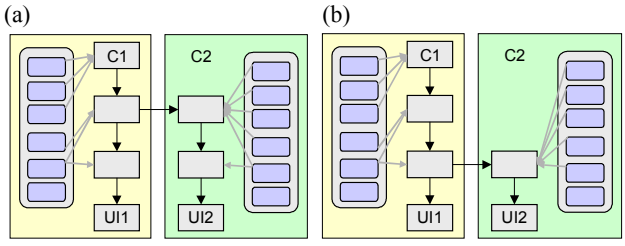

Figure 6. Translation at abstract (a) and concrete (b) UI levels.

- Fig. 6 depicts two other possible translations: at the abstract (Fig. 6a) and concrete UI level (Fig. 6b) respectively. For example, a mechanism is described in [15] to start from a same task model, but to derive different abstract UIs depending on parameters describing contexts of use. In a C1 context for instance, an abstract presentation will be derived that encompasses all information related to any combination of sub-tasks of the main task. In another C2 context, another abstract presentation may be generated, for instance with one presentation unit for each sub-task. The mechanism is based on the task tree configuration and on temporal operators between subtasks. Xweb supports cross-modality UIs for a same task [18]: different concrete UIs are produced from the same abstraction, but varying depending on available modalities and devices.
- Fig. 7a depicts the *ideal situation*: reification is applied until the very last step. Consistency is here minimally maintained since any high-level change can be propagated to the lower levels in a straightforward way. For example, this approach has been used for the Home Heating Control System for Java-enabled target platforms: UIs shown in Fig. 1 have been automatically generated using the ARTStudio development environment (Adaptation by Reification and Translation) [4,24]. Another example provided by [14] highlights an approach where HTML web pages are transformed on the fly into WML decks when a user accesses a non-WAP enabled web site via her cellular phone. This approach is mainly transcoding.

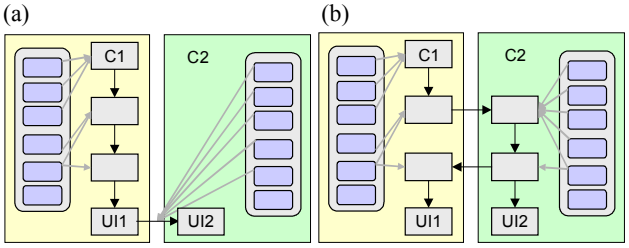

Figure 7. Reification before a final translation (a). Interleaving of reifications/translations (b).

- Fig. 7b shows a mix of interleaving between reification and translation. Such a combination makes it possible not to produce some intermediary models for a given context. Indeed the designer only has to translate to another context, generate them there by reification and then translate back to the initial context.

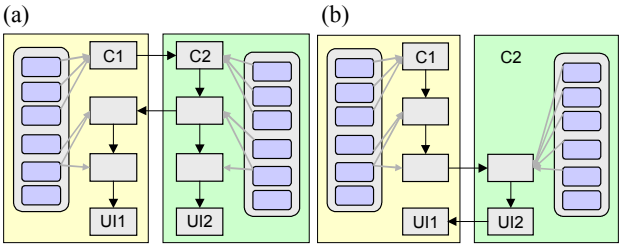

Figure 8. Interleaving of reifications/translations at highest (a) and lowest levels (b).

- The mix of interleaving between reification and translation may occur at any level, including the highest level (Fig. 8a) and the lowest level (Fig. 8b). For example, the combination of systems in [8] enables designers to transform task and domain models from one context to another and to produce an abstract UI for both. Lopez & Szekely [16] can compose different presentations for the Web and the Palm at the same time by classification of widgets and page recomposition.

NEEDS FOR REVISING THE FRAMEWORK

As we have seen in the previous section, the reference framework exhibits the capability of expressing many approaches for producing multi-target UIs [24]. However, some existing approaches cannot be easily represented on this framework, thus leading to some shortcomings:

- The reference framework only supports top-down approaches in a straightforward scheme. Any bottom-up approach or combination of top-down and bottom up steps cannot be represented.
- The framework mainly assumes that one builds one or many UIs from initial models to the final UIs via transient models (*forward engineering*). It does not support *reverse engineering* approaches where a new UI can be obtained by transforming an already existing one into a new one. For example, VAQUITA [2] regenerates a concrete UI from a HTML page to transform it for another context of use by successive translation and reification. For this purpose, an *abstraction* activity should be performed before, like a reverse arrow. This abstraction can be performed at the concrete (Fig. 9a) [2] or at the abstract UI level (Fig. 9b). We may even imagine to abstract at the highest level, but this would involve design recovery from code, which is hard to achieve. In the above cases, no concept and task models are used, thus meaning that not all models should be necessarily used.

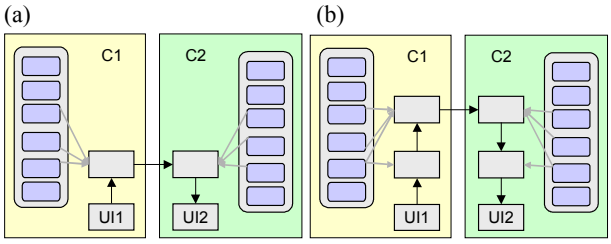

Figure 9. Regenerating a new UI from another one by abstraction at the concrete (a) or abstract (b) UI level.

- The framework also assumes that the only starting points are the models located at the highest or the lowest levels. This is not always true. For example, Girard & Baron [9] implemented a system that automatically generate UIs for different configurations from the definition of semantic functions contained in the semantic core. In this case, a concrete UI model is obtained for each context from an application model (Fig. 10a).

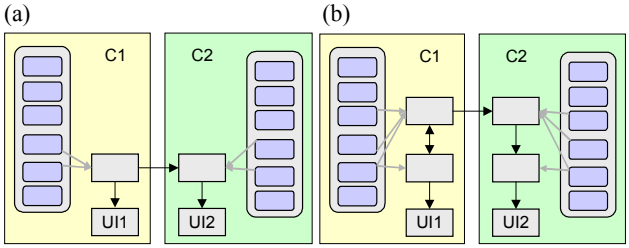

Figure 10. Generating concrete UIs from application model (a); Generating concrete UIs from many models (b).

- Moreover, the starting point may not always be unique: the application model is the starting model in [9], but it is the only one. Teallach [12] is a very representative example of a model-based approach [20] where the designer can start from any model: concept, task, or presentation. Here, multiple starting points can be considered, thus leading to many configurations (one of them is presented in Fig. 10b) where top-down and bottom-up arrows are nested. In particular, a designer may start with a presentation model, then derive a task model and link them together, with consistency with a domain model. Or she starts from a domain model, derives a task model, and a presentation model respectively.

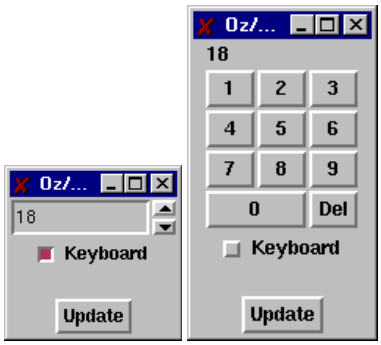

Figure 11. The numerical input with/out keyboard.

- Finally, the framework mainly represents development situations where the UIs are first analysed, and then designed to produce UIs. In this way, all potential UIs that can be accommodated by the approach, the development environment, and their supporting tools, are predefined. There is no way to express UIs that are known only at run-time. For example, Digestor [1] produces different UIs at runtime to display HTML web pages depending on constraints imposed by a computing platform. Similarly, there is no way to express runtime plastic UIs where the UIs are computed at runtime. For example, Grolaux [13] developed two UIs that cannot be described in terms of the framework: the numerical input (Fig. 11) automatically switches at run-time to a new configuration (here a virtual keyboard) when the keyboard is no longer available, even during task accomplishment. Similarly, FlexClock computes an appropriate concrete UI at run-time depending on the available screen space (Fig. 12).

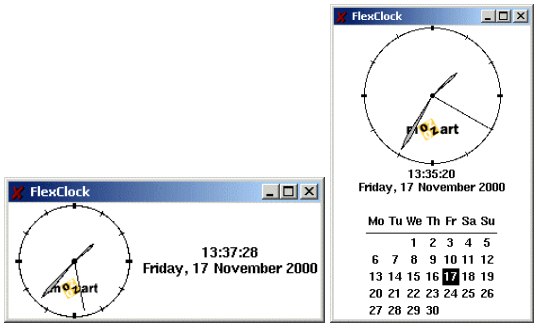

Figure 12. The plastic FlexClock.

By generalising what we observed so far, we can draw the following conclusions: not all models should be involved all the time, not all arrows should be top-down all the time, loops should be supported, multiple entry points should be allowed, multiple entry points can be considered simultaneously, any combination of multi-directional arrows should be supported. All of the principles that have been defined above are kept again in the revised reference framework. This new version, addressing the identified needs, is now introduced.

THE REVISED REFERENCE FRAMEWORK

The revised reference framework is a more general process that both refines the design time and extends its coverage to the run time.

Design time

At design time, the amendments are fourfold (Fig. 13):

- The notion of *entry point* is introduced. Entry point marks a level of reification at which the development process (reification and/or translation) may be began. Unlike the initial process that contains an entry point only (the task oriented specification), the revised reference framework foresees entry points at any level of reification: for example henceforward the designer may start at the abstract user interface without having produced the task oriented specification.
- At the *reification level* along a bottom-up perspective, a reverse engineering process make it possible to infer abstract models from more concrete ones. This amendment takes into account the practice consisting in directly prototyping the concrete user interface without having produced the task-oriented specification. From now on, these abstract specifications may be automatically and/or manually computed from the mockups.
- At the *translation level*, a cross operator is introduced to both translate to another context of use and change the level of reification. The crossing is a short cut that let possible to avoid the production of intermediary models.
- To sustain the runtime, a slice-based notation is introduced. It makes observable the life cycle of knowledge along the reification process: at level $I+p$, the slice I is mentioned if the knowledge modeled at I is still present at $I+p$. This traceability of knowledge may be useful for adaptation. For example in ARTStudio, as the task oriented specification is already lost at the abstract user interface, the user interface is not able to accomodate itself to another context of use. In consequence, an exterior supervisor has to manage the change of context by commuting from one pre-computed user interface to another one.

Based on this framework, the designer and/or the system may produce plastic user interfaces in a pre-computed way or on the fly at run-time.

Run-time

According to [4], the run-time process is structured as a three-step process: recognition of the situation, computation of a reaction, and execution of the reaction.

Situation Recognition

Recognising the situation includes the following steps:

- Sensing the context of use (e.g., current temperature is 22°C) by ad-hoc sensors and/or human skills;
- Detecting context changes (e.g., temperature increased from 18°C to 22°C) by ad-hoc mechanisms and/or human skills. Technology such as Aspect Oriented Programming or ad-hoc probes [4] can be helpful.
- Identifying context changes (e.g., for the heating control system, transition from the *regular* context to the *comfortable* context). Probabilistic approaches or fuzzy logic-based techniques could be applied.

In turn, the identification of context changes may trigger a reaction. There are two general types of trigger: entering a context and leaving a context. Schmidt suggests a third type of trigger [22], not considered in our discussion: being in a context. Triggers are combined with the AND/OR logic operators. For example, 'Leaving(C1) AND Entering(C2)' is a trigger that expresses the transition from Context C1 to Context C2. Having recognised the situation, the next step consists of computing the appropriate reaction.

Computation of a Reaction

The reaction is computed in the following way: Identify candidate reactions, select one of them, and apply the selected reaction.

- Identification of the candidate reactions. So far, we plan the following generic reactions:
  - Switch to another platform and/or to different environmental settings (e.g., switch from a portable PC to a PDA as the battery gets low, or turn the light on because the room grows dark).
  - Use another executable code: the current user interface is unable to cover the new context. It can't mould itself to the new situation and, in the meantime, preserve usability. Another executable code produced on the fly or in a pre-computed way is launched.
  - Adapt the user interface but keep the same executable code (e.g., switching from 1a to 1b when the screen gets too cluttered). The adaptation may be also pre-computed or computed on the fly.
  - Execute specific tasks such as turning the heat on. In this case, adaptation does not modify the presentation of the user interface, but it may impact the dialogue sequence.

  Some of these reactions may conserve or not the system state in terms of functional core adaptor and dialog controller. This persistence criteria may guide the selection among the candidate reactions.

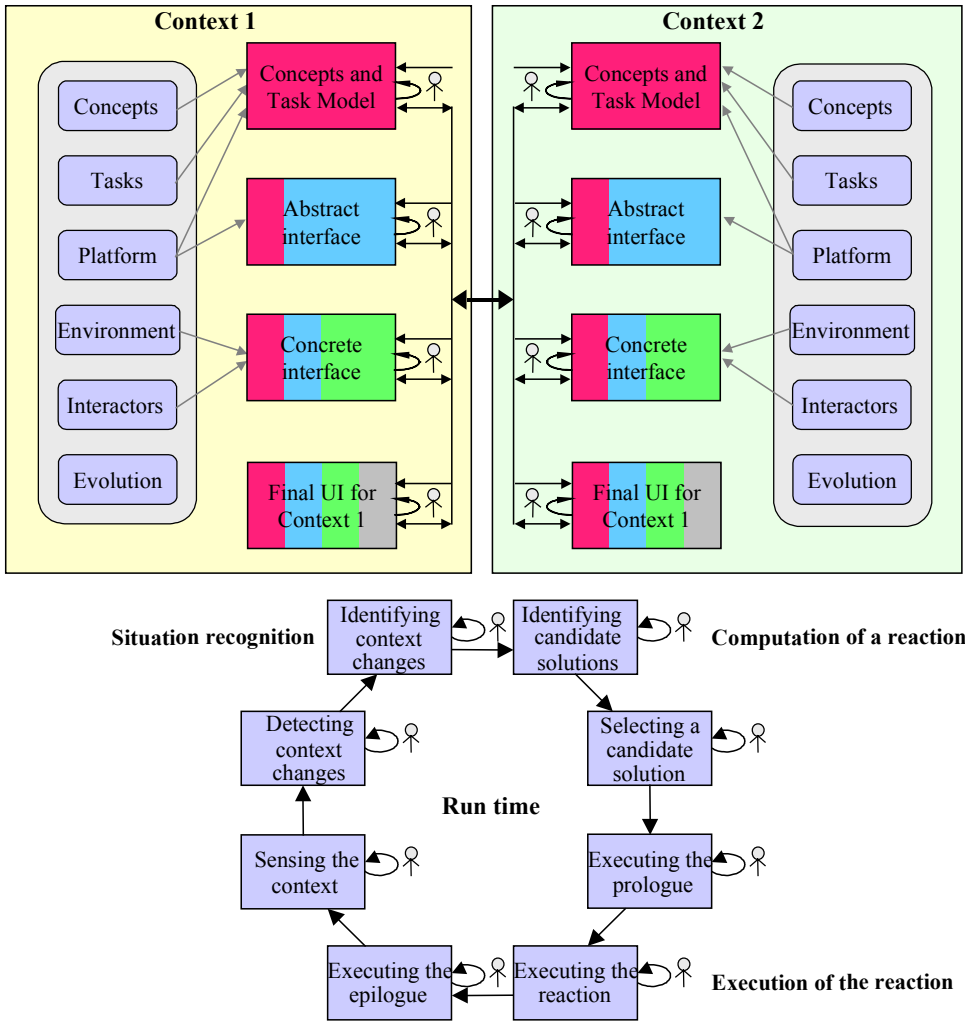

Figure 13. The revised reference framework.

- Selection of a candidate reaction according to an acceptable migration cost. Every reaction has a migration cost that expresses the effort the system and/or the user must put into this particular reaction. The effort is measured as a combination of criteria selected in the early phase of the development process.

Execution of the Reaction

The execution of the reaction consists of a prologue, the execution per se, and an epilogue:

- The prologue prepares the reaction. The current task is completed, suspended, or aborted; the execution context is saved (such as the specification of the temperature under modification); if not ready for use, the new version of the user interface is produced on the fly (e.g., a new presentation, a new dialogue sequence).
- The execution of the reaction corresponds to the commutation to the new version (e.g., the new presentation, the new dialogue sequence, or the execution of a specific task).
- The epilogue closes the reaction. It includes the restoration of the execution context (e.g., temperature settings, resuming of the suspended task).

Each one of the above steps is handled by the system, by the user, or by a co-operation of both. A step occurs on the fly or off-line. When a step is performed off-line, subsequent steps are also performed off-line. Transition between steps means transition between states. Transition between states has been analysed since the early developments of HCI. Norman's evaluation gap, Mackinlay's *et al.* use of graphical animation for transferring cognitive load to the perceptual level [17], the notion of visual discontinuity [11] etc., have all demonstrated the importance of transitions. A transition between two platforms, between executable codes, between UIs, etc. is therefore a crucial point that deserves specific research. The prologue and epilogue are here to help the designer thinking to the transitions.

CONCLUSION

Although the prospective development of interactive systems may be fun and valuable in the short run, we consider that the principles and theories developed for the desktop computer should not be put aside. Instead, our reply to the technological push is to use current knowledge as a sound basis, question current results, improve them, and invent new principles if necessary. This is the approach we have adopted for supporting plasticity by considering model-based techniques from the start. These techniques have been revised and extended to comply with a structuring reference framework. This framework has been put in practice both at design time and run time: ARTStudio provides a concrete, although incomplete, application of the design time aspect of the framework. On the contrary, *the Probe* [4] deals with the run time aspect for the detection of context changes. Now we focus on the evolution model to formalize the change of context and exploit it by an *Aspect Oriented Programming*.

ACKNOWLEDGMENTS

We thank Electricité de France, R&D Dept., for having supported the work conducted at CLIPS-IMAG.

REFERENCES

1. Bickmore, T.W., Schilit, B.N., *Digestor: Device-Independent Access To The World Wide Web*, in Proc. of 6th Int. World Wide Web Conf. WWW'6 (Santa Clara, April 1997), accessible at http://www.fxpal.com/PapersAndAbstracts/papers/bic97/
2. Bouillon, L., Vanderdonckt, J., Souchon, N., *Recovering Alternative Presentation Models of a Web Page with VAQUITA*, Chapter 27, in Proc. of 4th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2002 (Valenciennes, May 15-17, 2002), Kluwer Academics Pub., Dordrecht, 2002, pp. 311-322.
3. Calvary, G., Coutaz, J., Thevenin, D., *A Unifying Reference Framework for the Development of Plastic User Interfaces,* Proceedings of 8th IFIP International Conference on Engineering for Human-Computer Interaction EHCI'2001 (Toronto, 11-13 May 2001), R. Little and L. Nigay (eds.), Lecture Notes in Computer Science, Vol. 2254, Springer-Verlag, Berlin, 2001, pp. 173-192.
4. Calvary, G., Coutaz, J., Thevenin, D., *Supporting Context Changes for Plastic User Interfaces: a Process and a Mechanism*, in "People and Computers XV – Interaction without Frontiers", Joint Proceedings of AFIHM-BCS Conference on Human-Computer Interaction IHM-HCI'2001 (Lille, 10-14 September 2001), A. Blandford, J. Vanderdonckt, and Ph. Gray (eds.), Vol. I, Springer-Verlag, London, 2001, pp. 349-363.
5. Cockton, G., Clarke S., Gray, P., Johnson, C., *Literate Development: Weaving Human Context into Design Specifications,* in "Critical Issues in User Interface Engineering", P. Palanque & D. Benyon (eds), Springer-Verlag, London, 1995.
6. Eisenstein, J., Vanderdonckt, J., Puerta, A., *Model-Based User-Interface Development Techniques for Mobile Computing*, in Proc. of ACM Int. Conf. on Intelligent User Interfaces IUI'2001 (Santa Fe, January 14-17, 2001), J. Lester (ed.), ACM Press, New York, 2001, pp. 69-76.
7. Eisenstein, J., Rich, C., *Agents and GUIs from Task Models*, in Proc. of ACM Conf. on Intelligent User Interfaces IUI'2002 (San Francisco, Junary 13-16, 2002), ACM Press, New York, 2002, accessible at http://www-scf.usc.edu/~jeisenst/papers/iui02.pdf
8. Furtado, E., Furtado, J.J., Bezerra Silva, W., William Tavares Rodrigues, D., da Silva Taddeo, L., Limbourg, Q., Vanderdonckt, J., *An Ontology-Based Method for Designing Multiple User Interfaces*, ongoing submission.
9. Girard, P., Baron, M., *Construction interactive d'applications à partir du noyau fonctionnel*, in Proc. of Conférence sur l'Interaction Homme-machine et Ergonomie & Informatique ERGO-IHM'2000 (Biarritz, October 3-6, 2000), D.L. Scapin & E. Vergison (eds.), CRT ILS & ESTIA, Bidart, 2000, pp. 85-93.
10. Graham, T.C.N., Watts, L., Calvary, G., Coutaz, J., Dubois, E., Nigay, L., *A Dimension Space for the Design of Interactive Systems within their Physical Environments*, in Proc. of Conf. on Designing Interactive Systems DIS'2000 (New York, August 17-19, 2000,), ACM Press, New York, 2000, pp. 406–416.
11. Gram, C., Cockton, G. (eds.), *Design Principles for Interactive Software*, Chapman & Hall, London, 1996.
12. Griffiths, T., Barclay, P.J., Paton, N.W., McKirdy, J., Kennedy, J., Gray, P.D., Cooper, R., Goble, C.A., Pinheiro da Silva, P., Teallach: a model-based user interface development environment for object databases, Interacting with Computers, Vol. 4, No. 1, December 2001, pp. 31-68.
13. Grolaux, D., Van Roy, P., Vanderdonckt, J., *QTk: An Integrated Model-Based Approach to Designing Executable User Interfaces*, in PreProc. of 8th Int. Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'2001 (Glasgow, June 13-15, 2001), Ch. Johnson (ed.), GIST Tech. Report G-2001-1, Dept. of Comp. Sci., Univ. of Glasgow, Scotland, 2001, pp. 77-91. Accessible at http://www.dcs.gla.ac.uk/~johnson/papers/dsvis_2001/grolaux.
14. Kaasinen, E., Aaltonen, M. Kolari, J., Melakoski, S., Laakko, T., *Two Approaches to Bringing Internet Services to WAP Devices*, in Proc. of 9th Int. World-Wide-Web Conference WWW'9 (Amsterdam, 15-19 May 2000), accessible at http://www9.org/w9cdrom/228/228.html
15. Limbourg, Q., Vanderdonckt, J., Souchon, N., *The Task-Dialog and Task-Presentation Mapping Problem: Some Preliminary Results*, in Proc. of 7th Int. Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'2000 (Limerick, June 5-6, 2000), F. Paternò & Ph. Palanque (éds.), Lecture Notes in Computer Science, Vol. 1946, Springer-Verlag, Berlin, 2000, pp. 227-246.
16. Lopez, J.F., Szekely, P., *Web page adaptation for Universal Access*, in Proc. of Conf. on Universal Access in HCI UA-HCI'2001 (New Orleans, August 5-10, 2001), Lawrence Erlbaum Associates, Mahwah, 2001, pp. 690-694.
17. Mackinlay, J.D., Robertson, G.G, Card, S.K., *The Perspective Wall: Detail and Context Smoothly Integrated*, in Proc. of ACM Conference on Human Factors in Computing Systems CHI'91 (New Orleans, April 27 - May 2, 1991), ACM Press, New York, 1991, pp. 173-179.
18. Olsen, D.R., Jefferies, S., Nielsen, T., Moyes, W., Fredrickson, P., *Cross-Modal Interaction using Xweb*, Proc. of the ACM Symposium on User Interface Software and Technology UIST'2000 (San Diego, 6-8 November 2000), ACM Press, New York, 2000, pp.191-200.
19. Paternò, F., *Model-based Design and Evaluation of Interactive Applications*, Springer Verlag, Berlin, 1999.
20. Pinheiro da Silva, P., *User Interface Declarative Models and Development Environments: A Survey,* in Proc. of 7th Int. Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'2000 (Limerick, June 5-6, 2000), F. Paternò & Ph. Palanque (éds.), Lecture Notes in Comp. Sci., Vol. 1946, Springer-Verlag, Berlin, 2000, pp. 207-226.
21. Salber, D., Dey, A.K., Abowd, G. D., *The Context Toolkit: Aiding the Development of Context-Enabled Applications*, in Proceedings of ACM Conference on Human Factors in Computing Systems CHI'99 (Pittsburgh, 15-20 May 1999), ACM Press, New York, 1999, pp. 434-441.
22. Schmidt, A., *Implicit human-computer interaction through context*, Proc. of 2nd Workshop on Human Computer Interaction with Mobile Devices MobileHCI'01 (Edinburgh, 31 August 1999).
23. Thevenin, D., Coutaz, J., *Plasticity of User Interfaces: Framework and Research Agenda*, in Proc. of 7th IFIP International Conference on Human-Computer Interaction Interact'99 (Edinburgh, August 30 - September 3, 1999), Chapman & Hall, London, pp. 110-117.
24. Thevenin, D., *Adaptation en Interaction Homme-Machine: Le cas de la Plasticité*, Ph.D. thesis, Université Joseph Fourier, Grenoble, 21 December 2001.