

Supporting Context Changes for Plastic User Interfaces: A Process and a Mechanism

Gaëlle Calvary, Joëlle Coutaz & David Thevenin

CLIPS-IMAG, BP 53,38041 Grenoble Cedex 9, France

Tel: +33 476 514 854

Fax: +33 476 446 675

Email: {Gaelle.Calvary,Joelle.Coutaz,David.Thevenin}@imag.fr

URL: <http://iihm.imag.fr/>

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 to adapt to different contexts of use. We define a context of use as the set of values of variables that characterise the computational device(s) used for interacting with the system as well as the physical and social environment where the interaction takes place. A user interface is plastic if it is able to adapt to context changes while preserving usability. In this paper, we present a process and a software mechanism that support context changes for plastic user interfaces. We propose to structure adaptation as a three-step process: recognition of the situation, computation of a reaction to cope with the situation, and execution of the reaction. Reactions are specified in an evolution model which, in turn, is executed by a context supervisor. This supervisor is notified of context changes by a software probe that automatically detects deviations from the current situation. When notified, the supervisor executes the evolution model, and, when possible, adapts the user interface to the new context of use.

Keywords: human computer interaction, plasticity, adaptation, context of use, platform, environment.

1 Introduction

Recent years have seen the introduction of many types of access devices including Personal Digital Assistants (PDAs) and mobile phones (cf. Figure 1). Systems like CyberGuide (Abowd et al., 1996), the office assistant (Yan & Selker, 2000)



Figure 1: A wide variety of access devices.

and Welbo (Anabuki et al., 2000) all aim at providing the user with context-relevant information as the user moves around. New interaction techniques are being developed to support human tasks while hiding the computer away. These include transforming the PDA into a universal remote controller (Schilit et al., 1994), transferring data between PDAs by picking and dropping (Rekimoto, 2000), and augmenting real world objects with computational facilities (Bérard et al., 2000; Lee et al., 2000). Flexibility becomes more important: when the battery gets low, users may want to switch from their portable PC to the PDA without losing any contextual data. Similarly, they may want to switch from the PDA to a wall-sized electronic white board to share information in a more efficient way with a colleague passing by. These examples demonstrate that adaptation of interactive systems to context changes is becoming a major issue in HCI.

In this article, we address the problem of adaptation to context changes for plastic user interfaces (Thevenin & Coutaz, 1999; Calvary et al., to appear). We recall the definition of our notions of plasticity and context of use, then describe the process and a software mechanism we propose for supporting adaptation to context changes. We illustrate the discussion with the EDF home heating control system, a system we have developed according to our plasticity-related principles.

2 An Example: The EDF Heating Control System

A heating control system allows users to set the level of comfort in the home for different periods of the day. It also provides facilities for programming standard heating behaviour during weekends and vacations. The heating system is controlled with a dedicated wall-mounted device. EDF (the French Electricity Company) is willing to offer new interaction modalities using a variety of access devices suitable for different environments.

The various contexts of use envisioned by EDF include:

- At home, using a PDA connected to a wireless home-net.
- In the office, with a web server running on a standard workstation.
- Anywhere with a WAP-enabled mobile phone.

Clearly, the user interface of the heating control system cannot be the same for every access device: the variation of interactional resources requires specific solutions. Figure 2 shows three versions of the system:

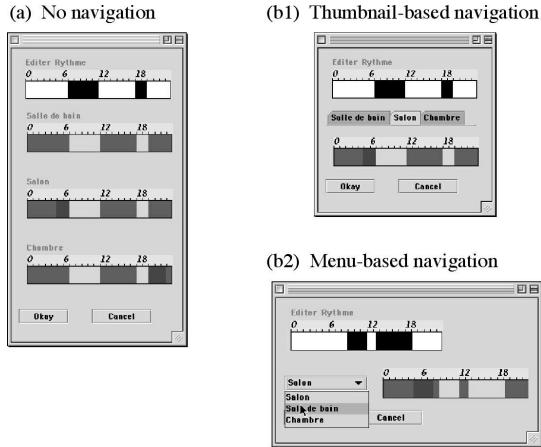


Figure 2: a) Large screen. The temperature of all the rooms of the home are available at a glance; b1 and b2) Small screen. The temperature of a single room is displayed at a time. A thumbnail or a menu allows the user to switch between rooms.

- In Figure 2a, the system displays the current temperature for each of the rooms of the house (the bedroom, the bathroom, and the living room). The screen size is large enough to make observable the entire system state. The user does not need to navigate between the rooms.
- In Figures 2b1 & b2, the screen is too small to display the whole system state. The temperature of a single room is shown at a time. In contrast with Figure 2a, the system state is not observable, but browsable (Gram & Cockton, 1996). As a result, a thumbnail must be introduced to navigate between the rooms (Figure 2b1), whereas in (Figure 2b2) navigation is based on a pull-down menu. The choice between the pull-down menu and a thumbnail depends on the shape of the screen as well as on the number of the rooms.

Figure 3 shows the interaction trajectory for setting the temperature of a room with a WAP-enabled mobile phone: On the left, the user selects the room (e.g. the living room — le salon). In the centre, the system shows the current temperature of the living room (i.e. 18°C). By selecting the editing function ('donner ordre'), the user is able to modify the current settings for that particular room (cf. the right most picture).

When compared to the situation depicted in Figure 2a, two navigation tasks (i.e. selecting the room, then selecting the edit function) must be performed to reach the desired state. In addition, a title must be added to every page to remind the user of the current location within the interaction space.

The user interfaces shown in Figures 2 & 3 have been produced with ARTStudio (Calvary et al., to appear), a tool for developing plastic user interfaces.



Figure 3: On mobile phones, screen size as well as the absence of direct manipulation require additional articulatory tasks: a navigation task to browse the rooms and a task to enter the edit mode.

3 Plasticity of User Interfaces

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 adapt to changes of the context of use while preserving usability” (Thevenin & Coutaz, 1999).

In this definition:

- A context of use is defined as the couple ‘Platform/Environment’ where *Platform* denotes the set of variables that characterise the computational device(s) used for interacting with the system. Typically, memory size, network bandwidth, screen size, etc. are determining factors. The *environment* covers the set of entities (e.g. objects, persons and events) that are peripheral to the current task(s) but that may impact the system and/or the user’s behaviour, either now or in the future. According to this definition, an environment may encompass the entire world. In practice, the boundary is set up by domain analysts whose role is to elicit the entities that are relevant to the case at hand. These include surrounding noise, lighting conditions, user’s and objects location, social ambience, etc.
- Adaptation is a reaction to context changes. Depending on the nature of the change, adaptation may consist of remodelling the user interface (as in Figure 2 when switching to a smaller screen). It may result in a task migration such as turning the heat on when the temperature is too low or hiding confidential information when someone gets too close to the owner’s screen. The adaptation is for the well being of the user, but is not targeted at the user’s current mental state. The user is supposed to have a predefined profile specified during the early phase of the development process.
- Usability is preserved if the properties elicited at the design stage for the particular system are kept within a predefined range of values. These

properties may be selected from general HCI properties such as those identified in (Gram & Cockton, 1996).

In summary, plasticity is a kind of adaptation. It results from a Situation \Rightarrow Reaction process where:

- Situation denotes a context change (of the platform state and/or of the environment) that requires a reaction.
- Reaction corresponds to the procedure that the system and/or the user executes to preserve usability.

In the next section, we focus the discussion on the adaptation process.

4 The Adaptation Process

Plastic adaptation is structured as a three-step process: recognition of the situation, computation of a reaction, and execution of the reaction.

4.1 Situation Recognition

Recognising the situation includes the following steps:

- Sensing the context of use (e.g. current temperature is 22°C).
- Detecting context changes (e.g. temperature has raised from 18°C to 22°C).
- Identifying context changes (e.g. for the heating control system, transition from the *regular* context to the *comfortable* context).

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 (200) suggests a third type of trigger, 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.

4.2 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.

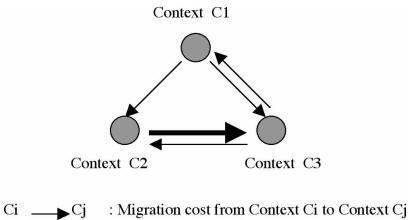


Figure 4: Graphical representation of migration costs between contexts. An arrow denotes the existence of a transition between two contexts. The thicker the arrow, the costlier is the transition.

- Adapt the user interface but keep the same executable code (e.g. switching from Figures 2a & b1 when the screen gets too cluttered).
- 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.
- 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. Figure 4 shows a graphical representation of migration costs between multiple contexts. An arrow denotes a potential transition. The thickness of an arrow expresses the cost. In the example of Figure 4, it is cheaper to switch from C1 to C2 than to migrate from C2 to C3. There is a potential transition from C1 to C2, but the reverse is impossible.
- The selected reaction is now applied.

4.3 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, etc.).
- 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 cooperation of both. A step occurs on the fly or offline. When a step is performed offline, subsequent steps are also performed offline. Transition between steps means transition between states. Transition between states has been analysed since the early developments of HCI. Norman's evaluation gap, Mackinlay et al.'s (1991) use of graphical animation for transferring cognitive load to the perceptual level, the notion of visual discontinuity (Gram & Cockton, 1996) etc., have all demonstrated the importance of transitions. A transition between two platforms, between executable codes, between user interfaces, etc. is therefore a crucial point that deserves specific research.

The process described so far applies whether the user, the system or a combination of both performs adaptation. In the rest of the paper, we concentrate the discussion on system-handled adaptation. For the reaction to be automatic, we suggest the implementation of the following concepts and mechanisms:

- The concept of a plasticity domain to identify the contexts a particular user interface is able to cover.
- An evolution model that specifies reactions to context changes.
- A context supervisor for handling adaptation.

These issues are discussed next in more detail.

4.4 Plasticity Domain and Plasticity Threshold

As defined above, a plastic user interface is able to adapt to different contexts of use while preserving usability. Figure 5 makes explicit the association of a platform with an environment to define a context of use. Given a particular application, the platforms and environments envisioned for this system are ranked against criteria computed from their attributes. For example, screen size, computational power and communication bandwidth, are typical attributes of the platforms planned for the EDF control system. Using these attributes, a PC would be ranked lower than a PDA since it imposes fewer constraints on the user interface. Similarly an environment with no noise would be ranked lower than an open area such as a street if speech recognition is planned as an interaction technique. Then, as shown in Figure 5:

- the *plasticity domain* of a user interface is the surface formed by all couples 'platform/environment' that this user interface is able to accommodate;
- the boundary of this surface defines the *plasticity threshold* of the user interface; and
- a *plasticity discontinuity* occurs when a change of context lies beyond this boundary.

For example, in Figure 5, Context C1 is covered by the user interface whereas Context C2 is outside its plasticity domain.

The model-based approach we apply for the development process of plastic user interfaces provides a sound framework for identifying the plasticity domain of a

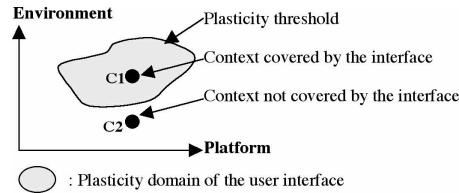


Figure 5: Context coverage and plasticity threshold of a user interface.

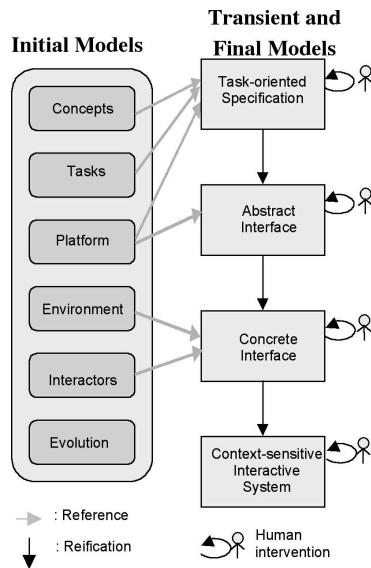


Figure 6: Development process of plastic user interfaces (Calvary et al., to appear).

particular user interface. As shown in Figure 6, our framework for the development of plastic user interfaces:

- builds upon known models such as the concepts and the task models; and
- introduces new models that have been overlooked or ignored to convey contexts of use: the platform, the interactors, the environment and the evolution models.

These models, which serve as input descriptions to the development process, are called ‘initial’. The process uses a combination of reification (vertical transformation shown in the picture) and translation (horizontal transformation not shown in the picture) to transform the initial models into transient models (i.e. the task-oriented description, the abstract and the concrete interfaces) until the final context-sensitive interactive system is produced.

The initial models can be referenced at any stage of the reification (translation) process. Delaying the reference to context models (i.e. the platform, environment, and interactors model) at later stages, results in a larger plasticity domain.

The evolution model, which is an initial model, has a specific role in the adaptation process. We describe it next.

4.5 The Evolution Model

The evolution model specifies the reaction to be applied when the context changes. For every triggering condition, the designer may specify a prologue, a reaction, and an epilogue. For example:

- When connecting to an Ethernet Network (triggering condition), set the system in ‘Office’ mode (reaction).
- When memory is tight, save the state of the running applications as well as the state of the working documents (prologue), reboot the system (reaction) then re-open the working documents (epilogue).

Prologues and epilogues are good locations for specifying transitions that have been tested to alleviate discontinuities between context changes. As mentioned above, the design of *articulatory user interfaces*, that is, portions of user interfaces dedicated to transitions between nominal situations, is an open issue.

The specifications of the prologue, as well as of the reaction and the epilogue, are optional:

- If no prologue is specified, no task is executed before the reaction.
- Similarly, if no epilogue is specified, no task is executed after the reaction *per se*.
- If no reaction is specified, the system computes a reaction depending on the case at hand (cf. Figure 7).

In Figure 7, consider Transition C1→C2 and the four possible situations:

- Case a: C1 and C2 belong to the plasticity domain of the current user interface. As a result, the current user interface is plastic enough to cover the new situation.
- Cases b and c: C1 and C2 belong to different plasticity domains. The current user interface does not apply any more. The set of user interfaces that cover C2 must be identified. In b there is one such candidate whereas in c and c’ we observe multiple target user interfaces, including the current user interface (cf. c’).
- Case d: C2 is not covered by any user interface. The user’s task is degraded, nay impossible to perform.

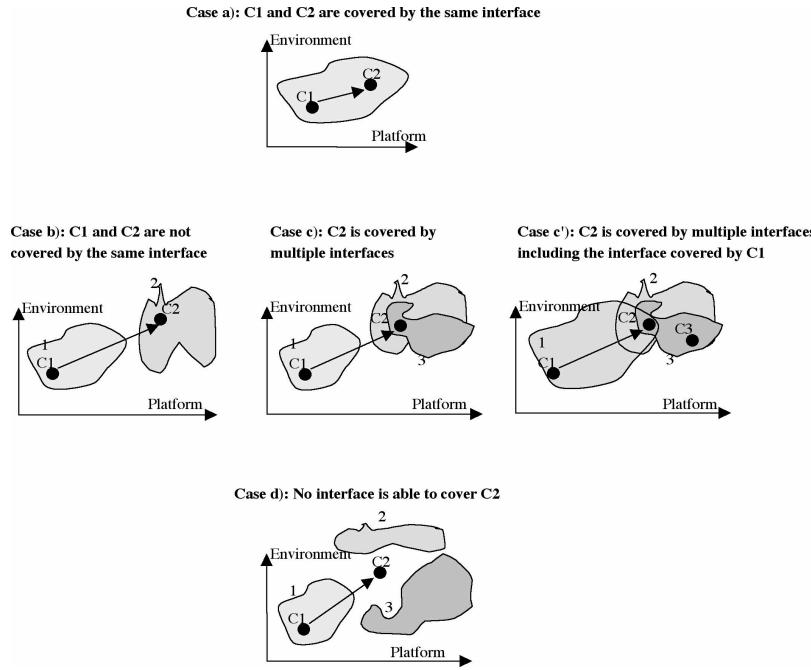


Figure 7: Four situations when migrating between contexts. In a) and c'), the current user interface remains valid; in b) and c) the user interface must be changed; in d) no user interface is able to cover the new context. In c) and c'), criteria must be used to decide between multiple target user interfaces.

When the new context is supported by multiple user interfaces (cases c and c'), criteria such as migration costs, must be brought to bear to guide the selection process. Case c' deserves additional comments: although the current user interface remains valid for C2, the adaptation process may decide to switch to a different user interface (e.g. that of C3) by anticipating further context transitions. Anticipation can be based on a history mechanism that records context transitions along with their migration costs as the user runs the system across multiple sessions. Considering the transition $C1 \rightarrow C2$ with Figure 7c', the target user interface may be selected according to the following rule: if the history mechanism indicates that the user is very likely to switch from C2 to C3, and if the migration cost from C2 to C3 is known to be high (as in Figure 4), and if there exists a target user interface U that covers both C2 and C3, then it is sensible to choose U when switching from C1 to C2.

In practice, the evolution model is not necessarily implemented as a centralised process. Instead, it may be distributed among the various user interfaces designed for the interactive system.

Our adaptation process structures the solution space for the problem of adapting user interfaces to context changes. In this space, a number of researchers are

concerned with the definition of the notion of context (Dey & Abowd, 2000; Salber & Abowd, 1998). Others are developing new sensing devices such as the Smart Floor (Orr & Abowd, 2000), smart localisers (Harter et al., 1999) and audio sensors (Clarkson et al., 1998) while software architecture models are being devised for capturing context at various levels of abstraction (Dey et al., 1999; Salber et al., 1999; Schmidt et al., 1999). Based on Salber's work (Salber et al., 1999), we are developing a programming model for capturing the context in terms of *contextors* (Rey, 2001). In the following section, we address the detection of context changes using a probe.

5 The Probe: a Software Mechanism for Detecting Context Changes

We have implemented a probe for automatically detecting deviations in user's performance and system properties (Calvary, 1998). This probe is part of CatchIt (Critic-based Automatic and Transparent tool for Computer-Human Interaction Testing) that combines both predictive and experimental approaches to usability testing. We are re-using the principles and mechanism of the CatchIt probe to detect context changes.

5.1 The CatchIt Probe

As a predictive tool, CatchIt verifies that general system properties such as observability, are satisfied by the system. As an experimental tool, CatchIt is able to detect whether the actual user's behaviour conforms to the designers' expectation. For doing so, designers specify *Situation* → *Reaction* rules that describe users' expected behaviours. For example, *If battery level gets low → switch to another platform*. From the rules specification:

- CatchIt predictively checks the observability of the concepts referred to in the *Situation* description. (In our example, if the battery level is not observable, there is little chance that the user will be aware of the situation).
- When the system is run with subjects, CatchIt experimentally checks whether users perform the expected actions (e.g. switching to another platform as the battery gets low).

In the current implementation of CatchIt, *situations* are expressed as Smalltalk statements. These statements make reference to the software objects that implement the concepts relevant to the situation. For example, *Platform Battery Level IsLow* is a statement that returns a boolean value to denote whether the current power level of the battery is low or sufficient.

Based on the *Situation* → *Reaction* specification, spy statements are automatically inserted in the source code of the application at the appropriate location. The mechanism is very similar to that of a debugger which modifies the application code without the programmer being aware of it.

Following up our example, suppose that:

- Class *Platform* has an attribute *Battery*.

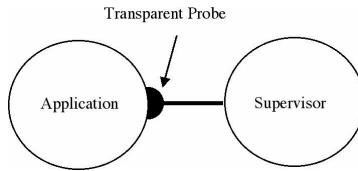


Figure 8: The context probe notifies the context supervisor with context events that may result in context changes.

- In turn, Class *Battery* has an attribute *Level*.
- Method *IsLow* of the class *Battery* compares *Level* with a class variable *Threshold*.

From the rule *Platform Battery Level IsLow → switch to another platform*, CatchIt is aware that Object *Level* of the battery must be traced. As a result, every method of the source code that has write access to a *Level* object is modified. Typically, write access methods include initialisation methods (*initialise* and *initialiseWith: aBattery*) and methods that modify an object attribute (for example *setLevel: anInteger* of the class *Battery*).

Spy statements, which are inserted at the very end of every write access method of traced objects, divert code execution and send trace events to a supervisor. A trace event contains the type of the write access method (i.e. creation, modification, or destruction), the method name being diverted and the traced object. In turn, the supervisor takes the appropriate action based on the *reaction* parts of the description rules.

Although CatchIt is intended for usability testing, the probe mechanism applies to the detection of context changes. Figure 8 illustrates the principle of the context probe.

5.2 The Probe for Detecting Context Changes

In the following discussion, we suppose that:

- Designers have modelled the context of use in terms of objects that denote the platform as well as the environment characteristics for the system at hand.
- Context is sensed and modelled as software objects at the appropriate level of abstraction as in (Salber et al., 1999).

As in CatchIt, write access methods to context objects are automatically augmented with spy statements that notify the context supervisor with context events (cf Figure 9). On receiving an event context, the supervisor infers the prologue, the reaction and the epilogue either:

- from the evolution model, when it exists; and/or

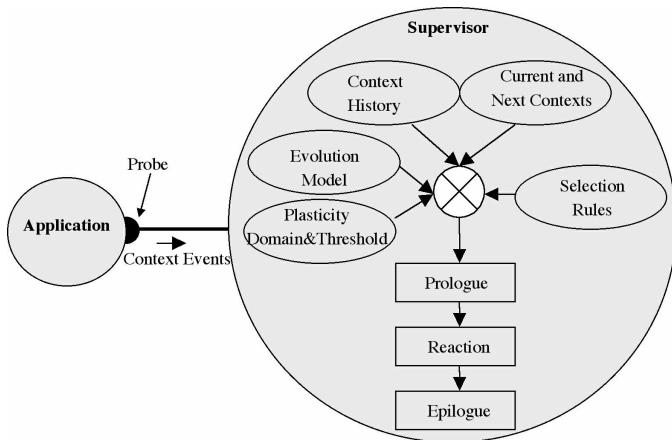


Figure 9: The adaptation process to context changes based on a transparent probe mechanism.

- from the plasticity domain of the current user interface coupled with selection rules and the history of previous contexts.

The supervisor then executes the selected prologue, the reaction and the epilogue. The context probe for plastic user interfaces is under implementation in Java using a *Listener* based approach.

6 Conclusion

This article extends our ongoing work on user interface plasticity. It focuses on the adaptation process and proposes a software mechanism that fits the general framework we are experimenting with for the development of plastic user interfaces. We show how adaptation can be supported at run time by a context supervisor coupled with a transparent context probe that automatically detects context changes. Adaptation is computed by the supervisor based on an evolution model and on the notions of plasticity domain, plasticity threshold and migration costs.

This work leaves opened a large number of issues including recommendations and heuristics for ranking environments and platforms. These are prerequisites for the computation of the plasticity domain and the plasticity threshold of a particular design solution. More importantly, research needs to be deployed in a systematic way to address seamless transitions between contexts.

Acknowledgement

This work has been supported by EDF, France.

References

- Abowd, G., Atkeson, C. G., Hong, J., Long, S., Kooper, R. & Pinkerton, M. (1996), Cyberguide: A Mobile Context-aware Tour Guide, Technical Report GIT-GVU-96-27,

GVU.

- Anabuki, M., Kabuka, H., Yamamoto, H. & Tamura, H. (2000), Welbo: An Embodied Conversational Agent Living in Mixed Reality Space, Videos of the 2000 Conference on Human Factors in Computing Systems (CHI'2000).
- Bérard, F., Coutaz, J. & Crowley, J. (2000), Le Tableau Magique: Un Outil pour l'Activité de Réflexion, in D. L. Scapin & E. Vergisson (eds.), *Actes de la Conférence ErgoIHM'2000*, CRT ILS & ESTIA, pp.33–40.
- Calvary, G. (1998), Proactivité et Réactivité: de l'Assignation à la Complémentarité en Conception et Evaluation d'Interfaces Homme-Machine, PhD thesis, Université Joseph-Fourier-Grenoble I, France.
- Calvary, G., Coutaz, J. & Thevenin, D. (to appear), A Unifying Reference Framework for the Development of Plastic User Interfaces, in R. Little & L. Nigay (eds.), *Proceedings of the 2001 Engineering of Human-Computer Interaction Conference (EHC1'2001)*, Lecture Notes in Computer Science, Springer-Verlag.
- Clarkson, B., Sawhney, N. & Pentland, A. (1998), Auditory Context Awareness via Wearable Computing, in M. Turk (ed.), *Proceedings of the 1998 Workshop on Perceptual User Interfaces (PUI'98)*, pp.37–42. <http://www.cs.ucsb.edu/PUI/PUIWorkshop98/Proceedings/Proc-Papers.html>.
- Dey, A. & Abowd, G. (2000), Towards a Better Understanding of Context and Context-awareness, in G. Szwilus, T. Turner, M. Atwood, B. Bederson, B. Bomsdorf, E. Churchill, G. Cockton, D. Crow, F. Détienne, D. Gilmore, H.-J. Hofman, C. van der Mast, I. McClelland, D. Murray, P. Palanque, M. A. Sasse, J. Scholtz, A. Sutcliffe & W. Visser (eds.), *Proceedings of the CHI 2000 Workshop on The What, Who, Where, When, Why and How of Context-awareness*, ACM Press, p.371.
- Dey, A., Salber, D., Futakawa, M. & Abowd, G. (1999), An Architecture To Support Context-Aware Applications, Technical Report GIT-GVU-99-23, GVU.
- Gram, C. & Cockton, G. (1996), *Design Principles for Interactive Software*, Chapman & Hall.
- Harter, A., Hopper, A., Steggles, P., Ward, A. & Webster, P. (1999), The Anatomy of a Context-aware Application, in *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99)*, ACM Press, pp.59–68.
- Lee, J., Su, V., Ren, S. & Ishii, H. (2000), HandSCAPE: A Vectorizing Tape Measure for On-site Measuring Applications, Videos of the 2000 Conference on Human Factors in Computing Systems (CHI'2000).
- Mackinlay, J. D., Robertson, G. G. & Card, S. K. (1991), The Perspective Wall: Details and Context Smoothly Integrated, in S. P. Robertson, G. M. Olson & J. S. Olson (eds.), *Proceedings of CHI'91: Human Factors in Computing Systems (Reaching through Technology)*, ACM Press, pp.173–9.
- Orr, R. & Abowd, G. (2000), The Smart Floor: A Mechanism for Natural User Identification and Tracking, Technical Report GIT-GVU-00-02, GVU.

- Rekimoto, J. (2000), Multiple Computer User Interfaces: “Beyond the desktop”, Direct Manipulation Environments, Videos of the 2000 Conference on Human Factors in Computing Systems (CHI’2000).
- Rey, G. (2001), Le Contexte en Interaction Homme–Machine, Diplom d’Etude Approfondies (DEA), Université Joseph Fourier–Grenoble I, France.
- Salber, D. & Abowd, G. (1998), The Design and Use of a Generic Context Server, in M. Turk (ed.), *Proceedings of the 1998 Workshop on Perceptual User Interfaces (PUI’98)*, pp.63–66. <http://www.cs.ucsb.edu/PUI/PUIWorkshop98/Proceedings/Proc-Papers.html>.
- Salber, D., Dey, A. & Abowd, G. (1999), The Context Toolkit: Aiding the Development of Context-enabled Applications, in M. G. Williams, M. W. Altom, K. Ehrlich & W. Newman (eds.), *Proceedings of the CHI99 Conference on Human Factors in Computing Systems: The CHI is the Limit*, ACM Press, pp.434–41.
- Schilit, B., Adams, N. & Want, R. (1994), Context-aware Computing Applications, in *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications (WMSCA’94)*, IEEE Computer Society Press, pp.85–90.
- Schmidt, A. (2000), “Implicit Human–Computer Interaction through Context”, *Personal Technologies* 4(2-3), 191–9. originally a paper in Brewster, S & Dunlop (eds), M, Proceedings of the 2nd Workshop on Human–Computer Interaction with Mobile Devices, p.23-7, 1999.
- Schmidt, A., Aidoo, K. A., Takaluoma, A., Tuomela, U., van Laerhoven, K. & van de Velde, W. (1999), Advanced Interaction in Context, in H.-W. Gellersen (ed.), *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing (HUC’99)*, Vol. 1707 of *Lecture Notes in Computer Science*, Springer-Verlag, pp.89–101.
- Thevenin, D. & Coutaz, J. (1999), Plasticity of User Interfaces: Framework and Research Agenda, in A. Sasse & C. Johnson (eds.), *Human–Computer Interaction — INTERACT ’99: Proceedings of the Seventh IFIP Conference on Human–Computer Interaction*, Vol. 1, IOS Press, pp.110–7.
- Yan, H. & Selker, T. (2000), Context-aware Office Assistant, in H. Lieberman (ed.), *Proceedings of 2000 International Conference on Intelligent User Interfaces (IUI 2000)*, ACM Press, pp.276–9.

Author Index

Calvary, Gaëlle, 1
Coutaz, Joëlle, 1

Thevenin, David, 1

Keyword Index

adaptation, 1

context of use, 1

environment, 1

human computer interaction, 1

plasticity, 1

platform, 1

