In Multi-Device and Multi-Context User Interfaces: Engineering and Applications Frameworks. Wiley Publ., H. Javahery Eds., 2003

A Reference Framework for the Development of Plastic User Interfaces

David Thevenin, Joëlle Coutaz, Gaëlle Calvary

CLIPS-IMAG,

BP 53, 38041 Grenoble Cedex 9, France

Contact author: Joelle.Coutaz@imag.fr

Abstract. This chapter addresses the plasticity of user interfaces from the software development perspective. We propose a conceptual framework that helps structure the development process of plastic user interfaces. This framework helps to clearly characterize the functional coverage of current tools such as ARTStudio and our own development environment for plastic user interfaces; it also helps to identify requirements for future tools.

Keywords. Context of use, Adaptation, Multi-target user interfaces, Plastic user interfaces, Context-sensitive user interfaces, software engineering

1 Introduction

Recent years have seen the introduction of a wide variety of computational devices including cellular telephones, personal digital assistants (PDA's), Internet enabled televisions (WebTV) and electronic whiteboards powered by high-end desktop machines. While the increasing proliferation of fixed and mobile devices responds to the need for ubiquitous access to information processing, this diversity offers new challenges to the HCI software community. These include:

- Constructing and maintaining versions of the user interface across multiple devices;
- Checking consistency between versions for ensuring a seamless interaction across multiple devices;
- Designing the ability to dynamically respond to changes in the environment such as network

connectivity, user's location, ambient sound and lighting conditions.

These requirements create extra cost in development and maintenance. In [Thevenin and Coutaz 1999], we presented a first attempt at cost-justifying the development process of user interfaces using the notion of plasticity as a foundational property for user interfaces. The term *plasticity* is inspired from 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 *contexts of use while preserving usability*" [Thevenin and Coutaz 1999].

Adaptation of user interfaces is a challenging problem. Although it has been addressed for many years [Thevenin 2001], these efforts have met with limited success. An important reason for this situation is the lack of a proper definition of the problem. In this chapter, we propose a reference framework that clarifies the nature of adaptation for plastic user interfaces from the software development perspective. It includes two complementary components:

- A taxonomic space that defines the foundational concepts and their relations for reasoning about the characteristics and software requirements of plastic user interfaces;
- A process framework that structures the software development of plastic user interfaces.

Our taxonomic space, called the "plastic UI snowflake" is presented in Section 3, followed in Section 4 by the description of the process framework. This framework is then illustrated in Section 5 with ARTStudio, a tool that supports the development of plastic user interfaces. In the following section, we introduce the terminology used in this chapter. In particular, we make clear the subtle distinction between plastic user interfaces and multi-target user interfaces in relation to context of use.

2 Terminology: Context of use, plastic UI and multi-target UI

Context is an all-encompassing term. Therefore, to be useful in practice, context must be defined in relation to a purpose. The purpose of this work is the adaptation of user interfaces to different elements that, combined, define a "context of use". Multi-targeting focuses on the technical aspects of user interface adaptation to different contexts of use. Plasticity provides a way to characterise system usability as adaptation occurs. These concepts are discussed next.

2.1 Context of use and target

The context of use denotes the run-time situation that describes the current conditions of use of the system. A target denotes a situation of use as intended by the designers during the development process of the system. More precisely:

The *context of use* of an interactive system includes:

- The people who uses the system;
- The platform used for interacting with the system;
- The physical environment where the interaction takes place.

A *target* is defined by:

- The class of users intended to use the system;
- The class of platforms that can be used for interacting with the system;
- The class of physical environments where the interaction is supposed to take place.

In other words, if at run-time the context of use is not one of the targets envisioned during the design phase, then the system is not able to adapt to the current situation (person, platform, physical environment).

A *platform* is modelled in terms of resources, which in turn determine the way information is computed, transmitted, rendered, and manipulated by users. Examples of resources include memory size, network bandwidth and input and output interaction devices. Resources motivate the choice of a set of input and output modalities and, for each modality, the amount of information made available. Typically, screen size is a determining factor for designing web pages. For DynaWall [Streitz et al. 1999], the platform includes three identical wall-size tactile screens mounted side by side. Rekimoto's augmented surfaces are built from a heterogeneous set of screens whose topology may vary: whereas the table and the electronic whiteboard are static surfaces, laptops may be moved around on top of the table [Rekimoto and Saitoh 1999]. These examples show that the platform is not limited to a single personal computer. Instead, it covers all of the computational and interaction resources available at a given time for accomplishing a set of correlated tasks.

An *environment* is "a set of objects, persons and events that are peripheral to the current activity but that may have an impact on the system and/or users' behaviour, either now or in the future" [Coutaz and Rey 2002]. According to our definition, an environment may encompass the entire world. In practice, the boundary is defined by domain analysts. The analyst's role includes observation of users' practice [Beyer 1998; Cockton et al. 1995; Dey et al. 2001; Johnson et al. 1993; Lim and Long 1994] as well as consideration of technical constraints. For example, environmental noise should be considered in relation to audio feedback. Lighting condition is an issue when it can influence the reliability of a computer vision-based tracking system [Crowley et al. 2000].

2.2 Multi-target user interfaces and plastic user interfaces

A multi-target user interface is capable of supporting multiple targets. A plastic user interface is a multi-target user interface that preserves usability across the targets. Usability is not intrinsic to a system. Usability must be validated against a set of properties elicited in the early phases of the development process. A multi-target user interface is plastic if these usability-related properties are kept within the predefined range of values as adaptation occurs to different targets. Although the properties developed so far in HCI [Gram and Cockton 1996] provide a sound basis for characterizing usability, they do not cover all aspects of plasticity. In [Calvary et al. 2001] we propose additional metrics for evaluating the plasticity of user interfaces.

Whereas multi-target user interfaces ensure technical adaptation to different contexts of use, plastic user interfaces ensure both technical adaptation and usability. Typically, portability of Java user interfaces supports technical adaptation to different platforms but may not guarantee consistent behaviour across these platforms.

2.3 Terminology: summary

In summary, for the purpose of our analysis:

- A target is defined as a triple "user, platform, environment" envisioned by the designers of the system,
- A context of use is a triple "user, platform, environment" that is effective at run-time,

- A multi-target user interface supports multiple targets, i.e., multiple types of users, platforms and environments. Multi-platform and multi-environment user interfaces are sub-classes of multi-target user interfaces;
- A multi-platform user interface is sensitive to multiple classes of platforms but supports a single class of users and environments;
- Similarly, a multi-environment user interface is sensitive to multiple classes of environments, but supports a single class of platforms and users. Multi-environment user interfaces are often likened to context-aware user interfaces [Moran and Dourish 2001];
- A plastic user interface is a multi-target user interface that preserves usability as adaptation occurs.

Having defined the notions of context of use, multi-target and plastic user interfaces, we are now able to present a taxonomic space that covers both multi-targeting and plasticity. The goal of this taxonomy is to identify the core issues that software tools aimed at multi-targeting and plasticity should address.

3 The "plastic UI snowflake"

Figure 1 is a graphic representation of the problem space for reasoning about user interface plasticity. The Plastic UI snowflake can be used for characterizing existing tools or for expressing requirements for future tools. Each branch of the snowflake makes explicit a number of issues relevant to UI plasticity. These include: the classes of targets that the tool supports (adaptation to platforms, environments and users), the stages of the development process that the tool covers (design, implementation or run-time), the actors that perform the adaptation of the user interface to the target (human or system intervention), the dynamism of user interfaces that the tools are able to produce (static pre-computed or dynamic on-the-fly computed user interfaces). When considering adaptation to multiple platforms, we also need to discuss the way the user interface migrates across platforms.

In the following sub-sections, we present each dimension of the snowflake in detail, illustrated with state-of-the-art examples. In particular, we develop multi-platform targeting. Although multi-user targeting is just as important, we are not yet in a position to provide a sound analysis for it. For adaptation to multi-environment targeting, please refer to [Moran and Dourish 2001] and [Coutaz and Rey 2002].

3.1 Target sensitivity

When considering software tools for plasticity, the first issue to consider is the kind of targets a particular tool addresses or is supposed to address. Are we concerned with multi-platform or multi-environment only? Do we need adaptation to multiple classes of users? Or is it a combination of platforms, environments and users?

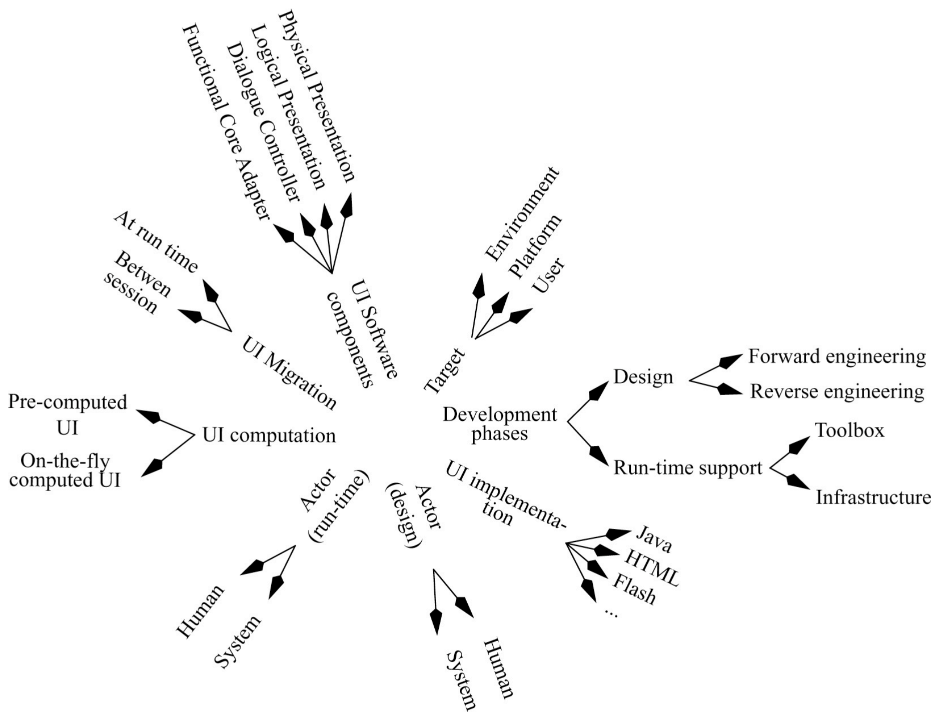


Figure 1. The *Plastic UI Snowflake*: a problem space for characterizing software tools, and for expressing requirements for software tools aimed at plastic user interfaces

For example, ARTStudio [Thevenin 2001] addresses the problem of multi-platform targeting whereas the Context Toolkit [Dey et al. 2001] is concerned with environment sensitivity only. AVANTI, which can support visually impaired users, addresses adaptation to end-users [Stephanidis et al. 2001]. We are not aware of any tool (or combination of tools) that supports all three dimensions of plasticity, i.e. users, platforms and environments.

3.2 Classes of software tools

As with any software tool, we must distinguish between tools that support the design phases of a system versus implementation tools and mechanisms used at run-time.

Design phases are primarily concerned with forward engineering and reverse engineering of legacy systems. Forward engineering is supported by specification tools for modelling, for configuration management and versioning, as well as for code generation:

- Modelling is a foundational activity in system design. In HCI, model-based tools such as Humanoid [Szekely 1996], ADEPT [Johnson et al. 1993] and TRIDENT [Vanderdonckt 1995], have shown significant promise, not only as conceptualisation tools, but as generators as well. If these approaches have failed in the past because of their high learning curve [Myers et al. 2000], they are being reconsidered for multi-target generation as in MOBI-D [Eisenstein et al. 2001] and USE-IT [Akoumianakis and Stephanidis 1997].
- Configuration management and versioning have been initiated with the emergence of large-scale software. They apply as well to multi-targeting and plasticity for two reasons. First, the code that supports a particular target can be derived from the high-level specification of a configuration. Second, the iterative nature of user interface development calls for versioning support. In particular, consistency must be maintained between the configurations that support a particular target;
- Generation has long been viewed as a reification process from high-level abstract description to executable code. For the purpose of multi-targeting and plasticity, we suggest generation by reification, as well as by translation where transformations are applied to descriptions while preserving their level of abstraction. The Process Reference framework described in Section 4 shows how to combine reification and translation;
- Tools for reverse engineering, such as eliciting software architecture from source code, are recent.
- In Section 4, we will see how tools such as Vaquita [Bouillon et al. 2002] can support the process of abstracting in order to “plastify” existing user interfaces.

Implementation phases are concerned with coding. Implementation may rely on infrastructure frameworks and toolkits. Infrastructure frameworks, such as the Internet or the X window protocol, provide implementers with a basic reusable structure that acts as a foundation for other system components such as toolkits. BEACH is an infrastructure that supports any number of display screens each connected to a PC [Tandler 2001]. MID is an infrastructure that extends Windows for supporting any number of mice to control a unique display [Hourcade and Bederson 1999]. We are currently developing I-AM (Interaction Abstract Machine), an infrastructure aimed at supporting any number of displays and input devices, which from the programmer's perspective will offer a uniform and dynamic interaction space [Coutaz and Lachenal 2002]. Similar requirements motivate the blackboard-based architecture developed for iRoom [Winograd 2001]. Dey et al.'s Context Toolkit is a toolkit for developing user interfaces that are sensitive to the environment [Dey et al. 2001].

3.3 Actors in charge of adaptation

The actors in charge of adaptation depend on the phase of the development process:

- At the design stage, multi-targeting and "plasticising" can be performed explicitly by humans such as system designers and implementers, and/or it can rely on dedicated tools;
- At run time, the user interface is adaptable or adaptive. It is adaptable when it adapts at the user's request, typically by providing preferences menus. It is adaptive when the user interface adapts on its own initiative. The right balance between adaptability and adaptivity is a tricky problem. For example, in context-aware computing, Cheverst et al. (2001) report that using location and time to simplify users' tasks sometimes makes users feel that they are being pre-empted by the system. Similarly, adaptivity to users has been widely attempted in the nineties with limited success [Browne et al. 1990].

3.4 Computation of multi-target and plastic user interfaces

The phases that designers and developers elicit for multi-targeting and plasticity have a direct impact on the types of user interfaces produced for the run time phase. Multi-target and plastic user interfaces may be pre-computed, and/or they may be computed on the fly:

- Pre-computed user interfaces result from adaptation performed during the design or implementation phases of the development process: given a functional core (i.e., an application), a specific user interface is generated for every envisioned target;

- Dynamic multi-target and plastic user interfaces are computed on the fly based on run-time mechanisms. Examples of run-time mechanisms include the Multimodal Toolkit [Crease et al. 2000], which supports dynamic adaptation to interaction devices. FlexClock [Grolaux 2000], which dynamically adapts to window sizes, is another example;
- The generated user interface can be a combination of static pre-computed components with on the fly adaptation. In this case, we have a hybrid multi-target plastic user interface. As a general rule of thumb, pre-computation is used for the overall structure of the user interface to ensure that the system runs quickly. However since this approach does not always provide an ideal adaptation to the situation, dynamic computation is added for fine-grain adjustments.

3.5 User interface software components

A number of software components are affected when adapting an interface for multi-targeting and plasticity. A large body of literature exists concerning this issue. However, because the software perspective is often mixed with the user's perception of adaptation, the state of the art does not provide a clear, unambiguous picture. For example, Dieterich et al. introduce five levels of adaptation: the lexical, syntactic, semantic, task and goal levels [Dieterich et al. 1993]. More recently, Stephanidis et al. define the lexical, syntactic and semantic levels of adaptation using examples as definitions [Stephanidis and Savadis 2001]. We propose to use Arch [Bass et al. 1992], a reference software architecture model, as a sound basis for characterizing software adaptation to target changes.

As shown in Figure 2, the Functional Core (FC) covers the domain-dependent concepts and functions. At the other extreme, the Physical Presentation Component (PPC), which is dependent on the toolkit used for implementing the look and feel of the interactive system, is in charge of presenting the domain concepts and functions in terms of physical interaction objects (also known as widgets or interactors). The keystone of the arch structure is the Dialogue Component (DC) whose role consists of regulating task sequencing. For example, the Dialogue Component ensures that the user executes the task "open document" before performing any editing task. FC, DC and PPC do not exchange data directly. Instead, they mediate through adaptors: the Functional Core Adaptor and the Logical Presentation Component.

The Functional Core Adaptor (FCA) is intended to accommodate various forms of mismatch between the Functional Core and the user interface. The Logical Presentation Component insulates the rendering of domain objects from the interaction toolkit of the target platform.

Using Arch as a structuring framework, the software components concerned by multi-targeting and plasticity are the FCA, the DC, the LPC, the PPC, or a combination of them. In particular:

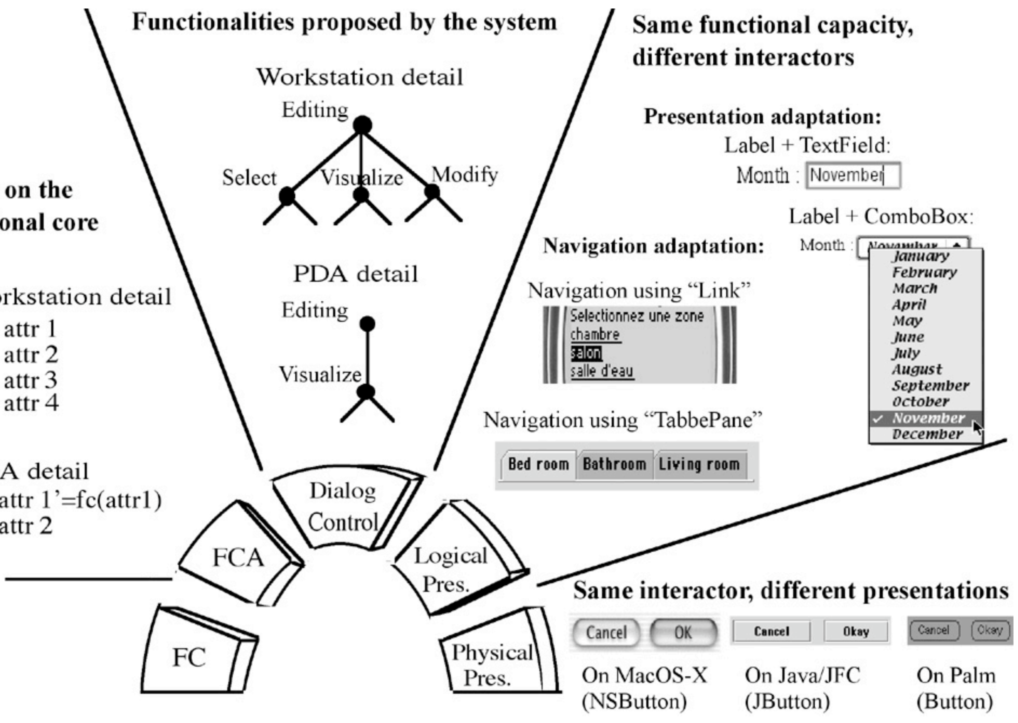


Figure 2. Arch architecture model

- At the Physical Presentation Component level, physical interactor classes used for implementing the user interface are kept unchanged but their rendering and behaviour may change across platforms. For example, if a concept is rendered as a button class, this concept will be represented as a button whatever the target platform is. However the look and feel of the button may vary. This type of adaptation is used in Tk as well as in Java/AWT with the notion of peers.
- At the Logical Presentation Component level, adaptation consists of changing the representation of the domain concepts. For example, the concept of month can be rendered as a Label+Textfield, or as a Label+Combobox, or as a dedicated physical interactor. In an LPC adaptation, physical interactors may change across platforms provided that their representational and interactional capabilities are

equivalent. The implementation of an LPC level adaptation can usefully rely on the distinction between Abstract Interactive Objects and Concrete Interactive Objects as presented in [Vanderdonckt and Bodard 1993].

- At the Dialog Component level, the tasks that can be executed with the system are kept unchanged but their organisation is modified. As a result, the structure of the dialogue structure is changed. AVANTI's polymorphic tasks [Stephanidis et al. 2001] are an example of a DC level adaptation.
- At the Functional Core Adaptor level, the nature of the entities as well as the functions exported by the functional core are changed. Zizi's semantic zoom is an example of an FCA level adaptation [Zizi and Beaudouin-Lafon 1994].

As illustrated by the above examples, Arch offers a clear analysis of the impact of a particular adaptation on the software components of a user interface.

3.6 User interface migration

User interface migration corresponds to the transfer of the user interface between different platforms. It may be possible either at run time or only between sessions:

- On-the-fly migration requires that the state of the functional core be saved as well as that of the user interface. The state of the user interface can be saved at multiple levels of granularity: when saved at the Dialogue Component level, the user can pursue the task from the beginning of the current task; when saved at the Logical Presentation or at the Physical Presentation levels, the user is able to carry on the current task at the exact point within the current task. There is no discontinuity.
- When migration is possible only between sessions, the user has to quit the application, and then restart the application from the saved state of the functional core. In this case, the interaction process is heavily interrupted. More research is required to determine how to minimize this disruption.

User interface migration between platforms puts high demands on the underlying infrastructure and toolkits. It also raises interesting user-centred design issues that should be addressed within the design process. Design phases are addressed next with the presentation of the Process Reference framework.

4 The Process Reference Framework for multi-target and plastic UIs

The "Process Reference Framework" provides designers and developers with generic principles for structuring and understanding the development process of multi-target and plastic user interfaces. We present an overall description of the framework in section 4.1 followed by a more detailed expression of the framework applied to the design stage in section 4.2. Different instantiations of the framework are presented in section 4.3. Run-time architecture, which can be found in [Crease et al. 2000] and [Calvary et al. 2001b], is not discussed in this chapter.

4.1 General description

As shown in Figure 3, the framework stresses a model-based approach coupled with a software development life-cycle.

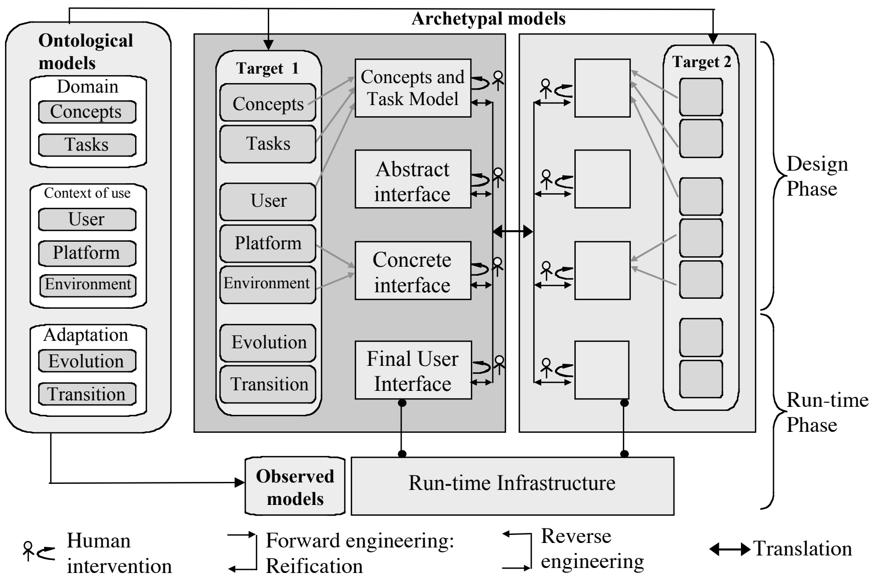


Figure 3. Process Reference Framework for the development of plastic user interfaces

4.1.1 *Models and life cycle*

Model-based approaches, which rely on high-level specifications, provide the foundations for code generation and code abstraction. This process of code generation and code abstraction reduces the cost of code production and code reusability while improving code quality.

The Process Reference Framework uses three types of models where each type corresponds to a step of the life cycle:

- *Ontological models* (left side of Figure 3) are meta-models that define the key dimensions of plasticity. They are independent from any domain and interactive system but are conveyed in the tools used for developing multi-target and plastic user interfaces. They are useful for the tool developer. When instantiated with tool support, ontological models give rise to archetypal models.
- *Archetypal models* depend on the domain and the interactive system being developed. They serve as input specifications for the design phase of an interactive system.
- *Observed models* are executable models that support the adaptation process at run-time.

As shown in Figure 3, the *design phase* complies with a structured development process whose end result is a set of executable user interfaces (Final User Interfaces) each aimed at a particular archetypal target.

4.1.2 Coverage of the models

As shown in Figure 3, The Process Reference Framework uses the following classes:

- *Domain models* cover the domain concepts and user tasks. Domain concepts denote the entities that users manipulate in their tasks. Tasks refer to the activities users undertake in order to attain their goals with the system.
- *Context of use models* describe a target in terms of user, platform and environment.
- *Adaptation models* specify how to adapt the system when the context of use and/or the target change. They include rules for selecting interactors, building user interface dialogue, etc.

These three classes of models (i.e., domain, context of use and adaptation models) may be ontological, archetypal or observed. As an illustration, in ARTStudio, the ontological task model is the ConcurTaskTree concepts [Breedvelt-Schouten et al. 1997] enhanced with decorations to specify the target for which each task makes sense. When instantiated as an archetypal task model, the ontological model can indicate that a given task does not make sense with a specific device and context, for example on a PDA in a train.

Having introduced the principles of the Process Reference Framework, we now present the framework as it is used in the design phase of multi-target and plastic user interfaces.

4.2 The Process Reference Framework in the design phase

In the design phase, the Process Reference Framework provides designers and developers with generic principles for structuring and understanding the development process of multi-target and plastic user interfaces. The design phase employs domain, context of use and adaptation models that are instantiations of the same models in the ontological domain. Archetypal models are referenced as well in the development process. As shown in Figure 3, the process is a combination of vertical reification and horizontal translation. Vertical reification is applied for a particular target while translation is used to create bridges between the descriptions for different targets. Reification and translation are discussed next.

4.2.1 Reification and translation

Reification covers the inference process from high-level abstract descriptions to run-time code. As shown in Figure 3, the framework uses a four-step reification process: a Concepts-and-Tasks Model is reified into an Abstract User Interface which, in turn, leads to a Concrete User Interface. The Concrete User Interface is then turned into a Final User Interface.

At the highest level, the *Concepts-and-Tasks Model* brings together the concepts and the tasks descriptions produced by the designers for that particular interactive system and that particular target.

An *Abstract User Interface* (Abstract UI) is a canonical expression of the rendering of the domain concepts and functions in a way that is independent of the interactors available for the target. For example, in ARTStudio, an Abstract UI is a collection of related workspaces. The relations between the workspaces are inferred from the task relationships expressed in the Concepts-and-Tasks model and from the structure of the concepts described in the Concept model. Similarly, connectedness between concepts and tasks is inferred from the Concepts-and-Tasks model. The canonical structure of navigation within the user interface is defined in this model as access links between workspaces.

A *Concrete User Interface* (Concrete UI) turns an Abstract UI into an interactor-dependent expression. Although a Concrete UI makes explicit the final look and feel of the Final User Interface, it is still a mock-up than runs only within the development environment.

A *Final User Interface* (Final UI), generated from a Concrete UI, is expressed in source code, such as Java and HTML. It can then be interpreted or compiled as a pre-computed user interface and plugged into a run-time infrastructure that supports dynamic adaptation to multiple targets.

A *translation* is an operation that transforms a description intended for a particular target into a description of the same class but aimed at a different target. As shown in Figure 3, translation can be applied between Task-and-Concepts for different targets, and/or between Abstract UI's, and/or Concrete UI's, and/or Final UI's.

Although high-level specifications are powerful tools, they have a cost. As observed by Myers et al. concerning the problem of "threshold and ceiling effects" [Myers et al. 2000], powerful tools require high learning curves. Conversely, tools that are easy to master do not necessarily provide the required support. Human intervention, decoration and factorisation, discussed next, help solve this dual problem.

4.2.2 Human Intervention

In the absence of tool support, reification and translation are performed manually by human experts. At the other extreme, tools can perform them automatically. But full automation has a price: either the tool produces common-denominator solutions (e.g., standard WIMP UI's produced by model-based UI generators), or the designer has to specify an overwhelming number of details to get the desired results.

As shown in Figure 3, the Process Reference Framework addresses cooperation between human and tool as follows: the development environment infers descriptions that the designer can then adjust to specific requirements. For examples, in ARTStudio, the designer can modify the relationships between workspaces, can change the layouts of the interactors, or even replace interactors. Decorations, presented next, provide another way to perform adjustments.

4.2.3 Decorations

A decoration is a type of information attached to description elements. Although a decoration does not modify the description per se, it provides information that modifies the interpretation of the description.

Applied to the design of multi-target and plastic UI, decorations are used to express exceptions to standard or default conditions. Designers focus on the representative target, and then express deviations from the reference case study. We propose three types of decorations:

- *Directive decorations* correspond to rules that cannot be easily expressed in terms of general-purpose inference rules. For example, suppose the development environment includes the following generation rule: "Any domain concept of type Integer must be represented as a Label in the Concrete UI". If the designer wishes to represent the temperature domain concept with a gauge, then a directive decoration can be attached to that concept. Directive decorations are pro-active. Corrective decorations are reactive.
- *Corrective decorations* can be used by designers to override standard options of the development environment. For example, suppose that, for workstations, the development environment generates the Final UI shown in Figure 5b. The designer can override the use of the thumbnail interactor, which gives access to one room at a time, by decorating the Concrete UI so as to obtain the presentation shown in Figure 5a, where all of the rooms are made observable at the same time.
- *Factorisation decorations* express exceptions to the standard case. They are useful for expressing local differences without modifying the model. They are complementary to the factorisation function presented next, which has a global result on the model.

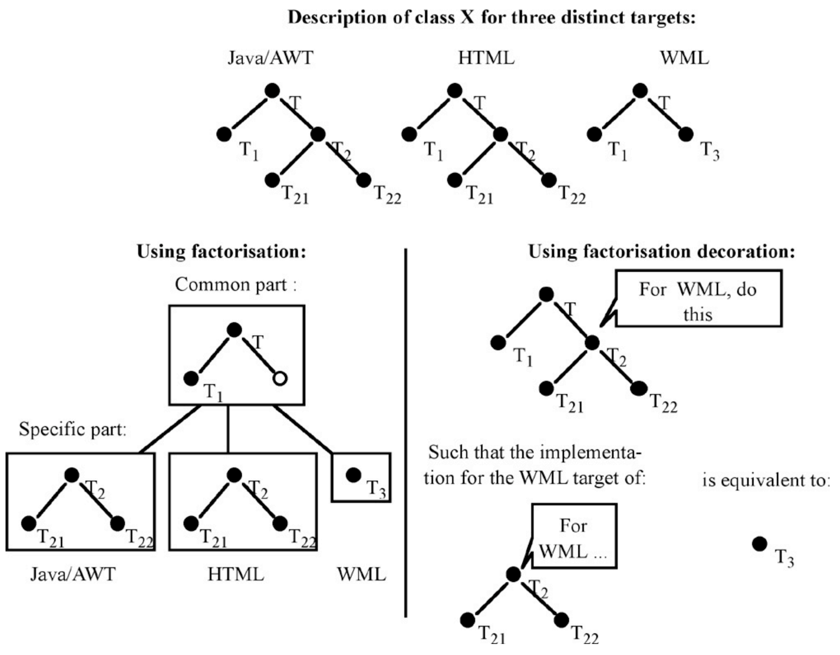


Figure 4. Factorisation function and factorisation decorations applied to task models

4.2.4 Factorisation

Figure 4 illustrates factorisation applied to task models. At the top are three task models for Java/AWT, HTML and WML-enabled target platforms. On the bottom left is the task model obtained from factorisation. On the bottom right is the task model obtained with factorisation decorations. Whereas factorisation produces 3 specific parts linked to a shared part, the use of factorisation description leads to a single shared description where exceptions are expressed as decorations.

Given a set of descriptions of the same class (e.g., a set of task models, of platform models, etc.), each aimed at a particular target, *factorisation* produces a new description composed of a description shared by all the targets and descriptions specific to each target. When applied to task models, factorisation corresponds to AVANTI's polymorphic tasks. The Figure 4 illustrates the principles of factorisation.

Factorisation supports multi-targeting and plasticity in a "one-target-at-a-time" approach. It allows designers to focus on one target at a time, followed by the combination of the descriptions produced independently for each target into a single description where common parts are factored out. Decoration supports a different approach where designers focus on a reference target and then define exceptions.

Having presented the general principles of the Process Reference Framework, we need to analyse how it can be instantiated.

4.3 Instantiations of the Process Reference Framework

The generic representation of the Process Reference Framework shown in Figure 3 can be instantiated in ways that reflect distinct practices and design cultures:

- Reification and translation can be combined in different patterns. For example ARTStudio uses reification all the way through the final user interface (Figure 7);
- The reification process can be started from any level of abstraction. Designers choose the *entry point* that best fits their practice. Typically, designers initiate the design by prototyping, i.e., by producing a concrete UI. If necessary, the missing abstractions higher in the reification process can be retrieved through reverse engineering as in Vaquita [Bouillon et al. 2002].
- The Process Reference Framework can be used to perform multi-targeting and plasticising through reverse engineering. Legacy systems, which have been designed for a particular target, must be redeveloped from scratch to support different targets. Alternatively, legacy systems can be reverse-engineered, then forward engineered. For example, an abstract UI can be inferred from a concrete UI. In turn, a concepts-and-tasks model can be retrieved from an abstract UI. This is the approach adopted in WebRevEnge in combination with Teresa [Mori et al. 2002].

The following section describes how our software tool ARTStudio implements a subset of the framework.

5 ARTstudio: An application of the Process Reference Framework

ARTStudio (Adaptation through Reification and Translation Studio) is a software environment for the development of pre-computed multi-platform UI’s. It supports elementary single screen platforms whose resources are known at the design stage. Therefore, distributed user interfaces are not addressed. It complies with the precepts of the Process Reference Framework, but implements a subset of its principles. Before discussing ARTStudio in more detail, we first present a case study: the EDF home heating control system.

5.1 The EDF home heating control system

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

- At home: through a dedicated wall-mounted device or through a PDA connected to a wireless home network;
- In the office: through the Web, using a standard workstation;
- Anywhere: using a WAP-enabled mobile phone.

Target users are archetypal adult family members, and the target environments are implicit.

A typical user's task consists of consulting and modifying the temperature of a room. Figures 5 and 6 show the final UIs of the system for the target platforms. In Figure 5a, the system displays the temperature settings for each of the rooms of the house (the bedroom, the bathroom, and the living room). Here 24-hour temperature settings are available at a glance for every room of the house. The screen size is comfortable enough to make the entire system state observable. In 5b, the system shows the temperature settings of a single room at a time. A thumbnail allows users to switch between rooms. Here 24-hour temperature settings are displayed for a single room at a time. In contrast with 5a, the system state is not observable, but is browsable [Gram and Cockton 1996] Additional navigational tasks, such as selecting the appropriate room, must be performed to access the desired information.

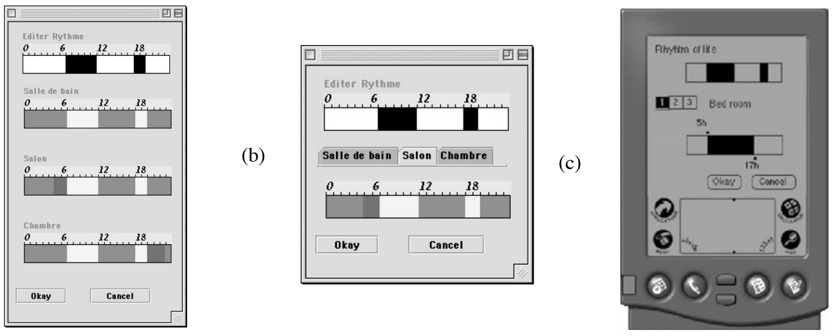


Figure 5. (a) Large screen; (b) Small screen; (c) PDA

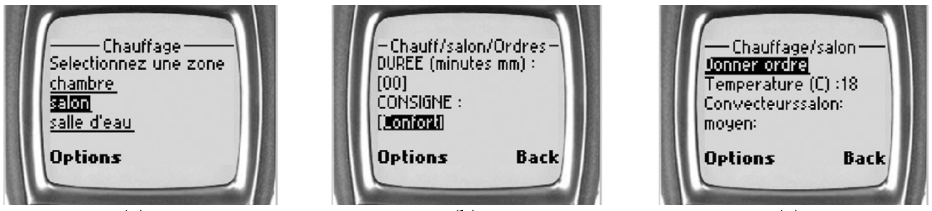
Figure 6 shows the interaction sequence for setting the temperature of a room with a WAP-enabled mobile phone. In 6a, the user selects the room (e.g., "le salon" – the living room). In 6b, the system shows the current temperature of the living room. By selecting the editing function ("donner ordre"), users can modify the temperature settings of the selected room (6c).

In comparison to the scenario depicted in Figure 5a, two navigation tasks (i.e., selecting the room, then selecting the edit function) must be performed in order to reach the desired state. In addition, a title has been added to every deck (i.e.WML page) to remind the user of their current location in the interaction space.

5.2 ARTStudio

In its current implementation, ARTStudio supports the four-step reification process of the Process Reference Framework, as well as human intervention and factorisation (Figure 7). It does not support translation or decoration. Multi-targeting is limited to Java-enabled single screen platforms and Web Pages using a Java HTTP server. ARTStudio is implemented in Java and uses JESS, a rule-based language, for generating abstract and concrete UIs. All of the descriptions used and produced by ARTStudio are saved as XML files.

In Figure 7, the first and second design steps are common to all targets. The difference appears when generating the concrete user interface. ARTStudio does not use translation.



Chauffage
Selectionnez une zone
chambre
salon
salle d'eau
Options

Chauff/salon/Ordres
DUREE (minutes mm) :
[00]
CONSIGNE :
[Confort]
Options Back

Chauffage/salon
Donner ordre
Temperature (C) :18
Convecteurssalon:
moyen:
Options Back

(a) (b) (c)

Figure 6. Modifying temperature settings using a WAP-enabled mobile phone

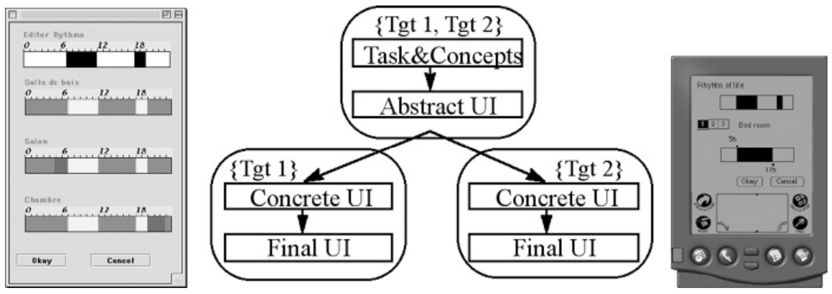


Figure 7. Instantiation of the Process Reference framework in ARTStudio

As illustrated in Figure 8, the development of a multi-platform plastic UI constitutes a project. A project includes the concepts-and-tasks model, the context of use model, the concrete UI and the final UI.

5.2.1 Domain concepts model

Domain concepts are modelled as UML objects using form filling or XML schema. In addition to the standard UML specification, a concept description includes the specification by extension of its domain of values. For example, the value of the attribute *name* of type *string* of the *room* concept may be one value among others including *Living room*, *Bed room*, etc. The type and the domain of values of a concept are useful information for identifying the candidate interactors in the Concrete UI. In our case of interest, the *room* concept may be represented as a set of strings (Figure 5a), as a thumbnail (Figure 5b), or as dedicated icons.

5.2.2 Task model

Task modeling is based on ConcurTaskTree [Breedvelt-Schouten et al. 1997] The ARTStudio task editor allows the designer to add, cut and paste tasks by direct manipulation. Additional parameters are specified through form filling. These include:

Specifying the name and the type of a task. For interaction tasks, the designer chooses from a predefined set of "universal" interaction tasks such as "selection", "specification" and "activation". This set may be extended to fit the work domain (e.g., "Set to anti-freeze mode").

- Specifying a prologue and an epilogue. This involves providing function names whose execution will be launched before and after the execution of the task. These functions are used when generating the final UI. They serve as gateways between the dialogue controller and the functional core adaptor of the interactive system. For example, for the task "setting the temperature of the bedroom", a prologue function is used to get the current value of the room temperature from the functional core. An epilogue function is specified to notify the functional core of the temperature change.

- Referencing the concepts involved in the task and ranking them according to their level of importance in the task. This ordering is useful for generating the abstract and concrete user interfaces: typically, first class objects should be observable whereas second class objects may be browsable if observability cannot be guaranteed due to the lack of physical resources.

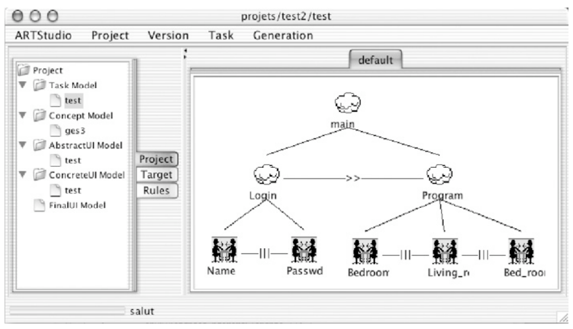


Figure 8. The task editor in ARTStudio. The picture shows the task model for a simplified version of the EDF Heating Control System

5.2.3 *Abstract UI model*

An abstract UI is modelled as a structured set of workspaces isomorphic to the task model: there is a one-to-one correspondence between a workspace and a task. In addition, a workspace that corresponds to an abstract task includes the workspaces that correspond to the subtasks of the abstract task: it is a compound workspace. Conversely, a leaf workspace is elementary. For example, Figure 9 shows three elementary workspaces (the bedroom, bathroom and living room) encapsulated in a common compound workspace. This parent workspace results from the Prog task of the task model. In turn, this workspace and the Rhythm elementary workspace are parts of the top-level workspace.

By direct manipulation, the designer can reconfigure the default arrangements. For example, given the task model shown at the top of Figure 10, the designer may decide to group the *Rhythm* workspace with the *Room* workspaces (Figure 10a) or, at the other extreme, suppress the intermediate structuring compound workspace (Figure 10b).

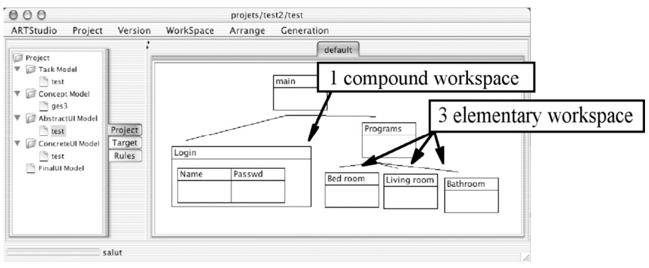


Figure 9. The Abstract UI generated by ARTStudio from the Task Model of Figure 8. Thick line rectangles represent compound workspaces whereas thin line rectangles correspond to elementary workspaces

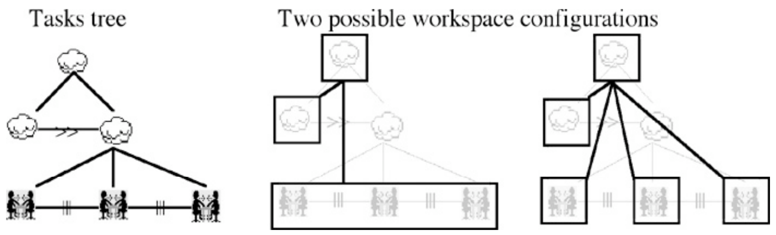


Figure 10. ARTStudio supports human intervention. Workspaces can be reconfigured at will

ARTStudio completes the workspace structure with a *navigation scheme* based on the logical and temporal relationships between the tasks. The navigation scheme expresses the user's ability to migrate between workspaces at run time.

5.2.4 The platform and interactor model

The platform model is a UML description that captures the size and depth of the screen, and the programming language supported by the platform (e.g., Java).

For each interactor available on the target platform, the interactor model specifies the representational capacity, interactional capacity, and the usage cost:

- Representational capacity: an interactor can either serve as a mechanism for switching between workspaces (e.g., a button, a thumbnail), or be used to represent domain concepts (e.g., a multi-valued scale as in Figure 5). In the latter case, the interactor model includes the specification of the data type it is able to render.
- The interactional capacity denotes the tasks that the interactor is able to support (e.g., specification, selection, navigation).

- The usage cost, which measures the system resources as well as the human resources the interactor requires, is expressed as the "x,y" footprint of the interactor on a display screen and its proactivity or reactivity (i.e., whether it avoids users to make mistakes *a priori* or *a posteriori* [Calvary 1998].

5.2.5 The concrete UI model

The generation of the concrete UI uses the abstract user interface, the platform and the interactor models, as well as heuristics. It consists of a set of mapping functions:

- Between workspaces and display surfaces such as windows and canvases;
- Between concepts and interactors;
- Between the navigation scheme and navigation interactors.

The root workspace of the Abstract User Interface is mapped into a window. Any other workspace is mapped either as a window or as a canvas depending on the navigation interactor used for entering the workspace. Typically, a button navigation interactor opens a new window whereas a thumbnail leads to a new canvas. Mapping concepts to interactors is based on a constraint resolution system. For each of the concepts, ARTStudio matches the type and the domain of values of the concepts with the interactors' representational capacity, the interactors' interactional capacity and their usage cost.

Figure 11 illustrates the concrete UIs that correspond to the final UI's shown in Figure 5a for large screen targets (e.g., Mac and PC workstations) and in Figure 5b for PDA targets.

Concrete UIs, like abstract UIs, are editable by the designer. The layout of the interactors can be modified by direct manipulation. In addition, the designer can override the default navigation scheme.

Pre-computed UIs can now be linked to a run-time environment that supports dynamic adaptation.

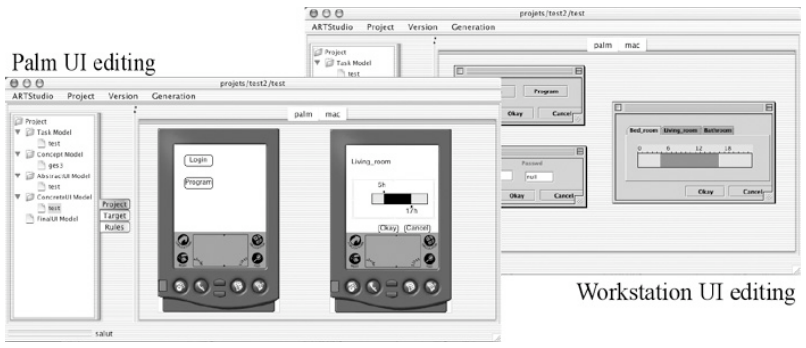


Figure 11. The concrete UI editor in ARTStudio

6 Conclusion

This chapter covers the problem of multi-target and plastic user interfaces from the software development perspective. A taxonomic space makes explicit the core issues that need to be addressed in the development and run-time stages of multi-target and plastic user interfaces. This taxonomy is complemented with a conceptual framework, the Process Reference Framework, which helps in structuring the development process of multi-target and plastic UIs as well as supporting run-time execution. The process reference framework helps to clearly characterise the functional coverage of current tools and identify requirements for future tools. Considerable research and development are needed to address the run-time infrastructure.

7 Acknowledgement

This work has been partly supported by the EC project IST CAMELEON (IST-2000-30104). The notions of entry point and abstraction have been suggested by Jean Vanderdonckt and Fabio Paternò, members of the CAMELEON project.

8 References

1. Akoumianakis, D. and Stephanidis, C. (1997) Supporting user-adapted interface design: The USE-IT system. *Journal Interacting with Computer, Elsevier,* 9, 73-104.
2. Bass, L., Little, R., Pellegrino, R., et *al.* (1992) The Arch model: Seeheim Revisited (version 1.0). The UIMS Developers Workshop (April 1991), published in *SIGCHI Bulletin*, 24 (1).
3. Beyer, H. and Holtzblatt K. (eds) (1998) *Contextual Design,* Morgan Kaufmann Publ.
4. Bouillon, L., Vanderdonckt, J. and Souchon, N. (2002) *Recovering Alternative Presentation Models of a Web Page with VAQUITA*. Chapter 27, Proceeding of 4th International Conference on Computer-Aided Design of User Interfaces CADUI, May 15-17, 2002, Valenciennes, France, Kluwer Academics Pub. Dordrecht.

5. Breedvelt-Schouten, I.M., Paternò, F.D. and Severijns, C.A. (1997) *Reusable structure in task models*. Proceeding of Eurographics Workshop on Design, Specification and Verification of Interactive System DSVIS, June 4-6, 1997, Granada, Spain. Springer Verlag.
6. Browne, D., Totterdell, P. and Norman, M. (eds) (1990) *Adaptive User Interface,* Academic Press, Computer and People Series.
7. Calvary, G. (1998) Proactivité et réactivité: de l'Assignation à la Complémentarité en Conception et Evaluation d'Interfaces Homme-Machine, Phd of the University Joseph-Fourier-Grenoble I, Speciality Computer Science.
8. Calvary, G., Coutaz, J. and Thevenin, D. (2001) *A Unifying Reference Framework for the Development of Plastic User Interfaces.* Proceedings of 8th IFIP International Conference on Engineering for Human-Computer Interaction EHCI, May 11-13, 2001, Toronto, Canada. Lecture Notes in Computer Science, Vol. 2254, Springer-Verlag.
9. Calvary, G., Coutaz, J. and Thevenin, D. (2001b) *Supporting Context Changes for Plastic User Interfaces: a Process and a Mechanism,* Proceeding of the joint AFHIM conference on Interaction Homme Machine and British HCI Group conference on Human Computer Interaction HCI, (eds A. Blandford, J. Vanderdonckt, P. Gray), BCS conference series, Springer Publ.
10. Cheverst, K., Davies, N., Mitchell, K. and Efstratiou, C. (2001) Using context as a Crystal Ball: Rewards and Pitfalls. *Journal on Personal and Ubiquitous Computing*, 5 (1), 8-11.
11. Cockton, G., Clarke S., Gray, P. and Johnson, C. (1995) Literate Development: Weaving Human Context into Design Specifications, in *Critical Issues in User Interface Engineering*, (eds P. Palanque and D. Benyon), Springer-Verlag, London Publ.
12. Coutaz, J., Lachenal, C. and Rey, (2002) G. Initial Reference Framework for Interaction Surfaces, Version V1.0, GLOSS Deliverable D17.
13. Coutaz, J. and Rey, G. (2002*) Foundations for a Theory of Contextors*. Proceeding of 4th International Conference on Computer-Aided Design of User Interfaces CADUI, May 15-17, 2002, Valenciennes, France, Kluwer Academics Pub. Dordrecht.
14. Crease, M., Gray, P. and Brewster, S. (2000) *A Toolkit Mechanism and Context Inde-pendent Widgets*. Proceeding of ISCE Workshop on Design, Specification and Verifica-tion of Interactive System DSVIS, June 5-6, 2000, Limerick, Ireland. Springer Verlag.
15. Crowley, J., Coutaz, J. and Bérard, F. (2000) Things that See. *Communication of the ACM*, 43 (3), 54-64.

16. Dey, A., Abowd, G. and Salber, D. (2001) A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Journal on Human Computer Interaction, Lawrence Erlbaum Publ.,* 16 (2-4), 97-166.

17. Dieterich, H., Malinowski, U., Kühme, T. and Schneider-Hufschmidt, (1993) M., State of the Art in Adaptive user Interfaces, in *Adaptive User Interfaces, Principles and Practice*. (eds H.J. Bullinger, P.G. Polson) Elsevier Publ., Human Factors in Information Technology series.

18. Eisenstein J., Vanderdonckt J. and Puerta A. (2001) *Applying Model-Based Techniques to the Development of UIs for Mobile Computers*. Proceeding of ACM Conference on Intelligent User Interfaces IUI, Junary 14-17, 2001, Santa Fe, New Mexico, USA, ACM Press.

19. Gram, C. and Cockton, G. (eds) (1996) *Design Principles for Interactive Software*. Chapman & Hall.

20. Grolaux, D., (2000) FlexClock. Accessible on the Internet Web Site http://www.info.ucl.ac.be/people/ned/flexclock/

21. Hourcade, J. and Bederson, B. (1999) Architecture and Implementation of a Java Package for Multiple Input Devices (MID). Available at: http://www.cs.umd.edu/hcil/mid/

22. Johnson, P. Wilson, S., Markopoulos, P. and Pycock, Y. (1993) *ADEPT-Advanced Design Environment for Prototyping with Task Models*. Proceedings of the joint ACM Conference on Human Factors in Computing Systems CHI and IFIP Conference on Human Computer Interaction INTERACT, April 24-29, 1993, Amsterdam, The Netherlands, ACM Press.

23. Lim, K. Y. and Long, J. (eds) (1994) *The MUSE Method for Usability Engineering*. Cambridge Univ. Press.

24. Moran, T. and Dourish, P, (eds) (2001) *Context-Aware Computing*, Special Issue of Human Computer Interaction, Lawrence Erlbaum, 16 (2-4).

25. Mori, G., Paganelli, L., Paternò, F., et *al.* (2002) Tools for Model-Based Design of Multi-Context Applications September 5, 2002 CAMELEON Document, Deliverable 2.1.

26. Myers, B., Hudson, S. and Pausch, R. (2000) Past, Present, Future of User Interface Tools. *Transactions on Computer-Human Interaction*, 7 (1), 3–28.

27. Rekimoto, J. and Saitoh, M. (1999) *Augmented Surfaces: A spatially continuous workspace for hybrid computing environments*. Proceeding of the ACM conference on Human Factors in Computing Systems CHI, May 15-20, 1999, Pittsburgh, PA, USA. ACM Press.

28. Stephanidis, C., Paramythis, A., Sfyrakis, M. and Savidis, A. (2001) A case study in Unified User Interface Development: The AVANTI Web Browser, in *User Interface for All, Concepts, methods and tools*, (eds C. Stephanidis), Lawrence Erlbaum, Publ.

29. Stephanidis, C. and Savidis, A. (2001) Universal Access in the Information Society: Methods, Tools, and Interaction Technologies. *Journal of the Universal Access in Information Society UAIS*, 1 (1), 40-55.

30. Streitz, N. Geißler, J., Holmer, T. et *al.* (1999) *I-LAND: An interactive landscape for creativity and innovation*. Proceeding of the ACM conference on Human Factors in Computing Systems CHI, May 15-20, 1999, Pittsburgh, PA, USA. ACM Press.

31. Szekely P. (1996) *Retrospective and Challenges for Model-Based Interface Development*. Proceeding of the 2nd International Workshop on Computer-Aided Design of User Interfaces CADUI, June 5-7, 1996. Namur, Belgium. (eds J. Vanderdonckt), Presses Universitaires de Namur.

32. Tandler, P. (2001) *Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices*. Proceeding of 3rd ACM conference on Ubiquitous Computing Ubicomp. September 30 – October 2, 2001. Altanta, Georgia, USA, Springer Verlag.

33. Thevenin, D. (2001) Adaptation en Interaction Homme-Machine : le cas de la Plasticité. Phd Thesis of the University Joseph-Fourier Grenoble I*,* Speciality Computer Science.

34. Thevenin, D. and Coutaz, J. (1999) *Plasticity of User Interfaces: Framework and Research Agenda.* Proceeding of IFIP Conference on Human Computer Interaction INTERACT. August 30 – September 3, 1999. Edinburgh, Scotland. Sasse, A. and Johnson, C. (eds), IFIP IOS Press.

35. Vanderdonckt, J. (1995) Knowledge-Based Systems for Automated User Interface Generation; The TRIDENT Experience. RP-95-010, University N-D de la Paix, Computer Science Institute, Namur B.

36. Vanderdonckt, J. and Bodard, F. (1993) *Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection*. Proceedings of the joint ACM Conference on Human Factors in Computing Systems CHI and IFIP Conference on Human Computer Interaction INTERACT, April 24-29, 1993, Amsterdam, The Netherlands, ACM Press.

37. Winograd, T. (2001) Architecture for Context. *Journal on Human Computer Interaction, Lawrence Erlbaum Publ*, 16 (2-4), 401-419.

38. Zizi, M. and Beaudouin-Lafon, M. (1994) *Accessing Hyperdocuments through Interactive Dynamic Maps*. Proceeding of the European Conference on Hypertext Technology ECHT, September 19 – 23, 1994. Edinburgh, Scotland. ACM Press.