A unifying reference framework for multi-target user interfaces

Gaëlle Calvary*[,1], Joëlle Coutaz[1], David Thevenin[2], Quentin Limbourg[3], Laurent Bouillon[3], Jean Vanderdonckt[3]

[1]*IIHM - Laboratoire CLIPS, Fédération IMAG - BP 53, F-38041 Grenoble Cedex 9, France*
[2]*National Institute of Informatics, Hitotsubashi 2-1-2 1913, Chiyoda-ku, Tokyo, 101-8430, Japan*
[3]*Université catholique de Louvain, School of Management (IAG), Place des Doyens, 1 – B-1348 Louvain-la-Neuve, Belgium*

Abstract

This paper describes a framework that serves as a reference for classifying user interfaces supporting multiple targets, or multiple contexts of use in the field of context-aware computing. In this framework, a context of use is decomposed in three facets: the end users of the interactive system, the hardware and software computing platform with which the user have to carry out their interactive tasks and the physical environment where they are working. Therefore, a context-sensitive user interface is a user interface that exhibits some capability to be aware of the context (context awareness) and to react to changes of this context. This paper attempts to provide a unified understanding of context-sensitive user interfaces rather than a prescription of various ways or methods of tackling different steps of development. Rather, the framework structures the development life cycle into four levels of abstraction: task and concepts, abstract user interface, concrete user interface and final user interface. These levels are structured with a relationship of reification going from an abstract level to a concrete one and a relationship of abstraction going from a concrete level to an abstract one. Most methods and tools can be more clearly understood and compared relative to each other against the levels of this framework. In addition, the framework expresses when, where and how a change of context is considered and supported in the context-sensitive user interface thanks to a relationship of translation. In the field of multi-target user interfaces is also introduced, defined, and exemplified the notion of plastic user interfaces. These user interfaces support some adaptation to changes of the context of use while preserving a predefined set of usability properties.

Keywords: Context-aware computing; Context of use; Context-sensitive user interfaces; Model-based approach, Multi-platform user interfaces; Multi-target user interfaces; Plasticity of user interfaces

[1] Corresponding author. Tel.: +33-4-76.51.48.54; fax: +33-4-76.44.66.75.
E-mail addresses: Gaelle.Calvary@imag.fr, Joelle.Coutaz@imag.fr, thevenin@nii.ac.jp, limbourg @isys.ucl.ac.be, bouillon@isys.ucl.ac.be, vanderdonckt@isys.ucl.ac.be

1. Introduction

Recent years have seen the introduction of many types of computers, devices and Web appliances. In order to perform their tasks, people now have available a wide variety of computational devices ranging over an important spectrum: mobile phones, UMTS phones, smart and intelligent phones, Personal Digital Assistants (PDAs), pocket PC, handheld PC, Internet enabled televisions (WebTV), Internet ScreenPhones, Tiqit computers, interactive kiosks, tablet PCs, laptop or notebooks, desktops, and electronic whiteboards powered by high end desktop machines (Fig. 1). While this increasing proliferation of fixed and mobile devices fits with the need for ubiquitous access to information processing, this diversity offers new challenges to the HCI software community (Eisenstein et al., 2001). These include:

- Constructing and maintaining versions of single applications across multiple devices.
- Checking consistency between versions for guaranteeing a seamless interaction across multiple devices.
- Building into these versions the ability to dynamically respond to changes in the environment such as network connectivity, user's location, ambient sound or lighting conditions.

To address these new requirements, the notions of *multi-targeting* and *plasticity* are introduced. Both deal with adaptation to *context of use* to cover this new diversity without exploding the cost of development and maintenance. We first define the notion of context of use and then precise the way multi-targeting and plasticity follow to satisfy these new requirements.

Fig. 1. The spectrum of available computing platforms.

1.1. Context of use

Context is an all-embracing term. Composed of "con" (with) and "text", context refers to the meaning that must be inferred from the adjacent text. As a result, to be operational, context can only be defined in relation to a purpose, or finality (Crowley et al., 2002). For the purpose of this study, the *context of use* of an interactive system is defined by three classes of entities:

- The users of the system who are intended to use (and/or who effectively use) the system,

- The hardware and software platform(s), that is, the computational and interaction device(s) that can be used (and/or are used, effectively) for interacting with the system,
- The physical environment where the interaction can take place (and/or takes place in practice).

The *user* represents a stereotypical user of the interactive system. For example, according to the Human Processor Model (Card et al., 1983), it may be described by a set of values characterizing the perceptual, cognitive and action capacities of the user. In particular, his perceptual, cognitive and action disabilities may be expressed in order to choose the best modalities for the rendering and manipulation of the interactive system.

The *platform* is modeled 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 for 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 an 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). In Pebbles, PDA's can be used as input devices to control information displayed on a wall-mounted electronic board (Myers et al., 1998). In Kidpad (Benford et al., 2000), graphics objects, which are displayed on a single screen, can be manipulated with a varying number of mice. 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.

The *environment* denotes "the 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 behavior, 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 set up by domain analysts whose role is to elicit the entities that are relevant to the case at hand. These include observation of users' practice (Beyer and Holzblatt, 1998; Cockton et al., 1995, Dey et al., 2001; Johnson et al., 1993; Lim and Long, 1994) as well as consideration for technical constraints. For example, surrounding noise should be considered in relation to sonic feedback. Lighting condition is an issue when it may influence the robustness of a computer vision-based tracking system (Crowley et al., 2000).

Based on these definitions, we introduce the notions of multi-targeting and plasticity that both deal with the diversity of contexts of use by adaptation.

1.2. Multi-targeting and plasticity

Multi-targeting and plasticity both address the diversity of contexts of use by adaptation. Whereas multi-targeting focuses on the technical aspects of adaptation, plasticity provides a way to qualify system usability as adaptation occurs. A *multi-target user interface* is capable of supporting multiple contexts of use. But no requirement is expressed in terms of usability. In the opposite, by analogy with the property of materials that expand and contract under natural constraints without breaking, a *plastic user interface* is a multi-target user interface that preserves usability as adaptation occurs. Usability is expressed as a set of properties selected in the early phases of the development process. It may refer to the properties developed so far in HCI (Gram and Cockton, 1996). According to these definitions, plasticity can be viewed as a property of adaptation. Consequently, we focus in this paper on multi-targeting. As the definition of multi-targeting relies on the multiplicity of contexts of use, the granularity of a context has to be clarified. Under software engineering considerations, we distinguish two kinds of contexts of use:

- Predictive contexts of use that are foreseen at design time.
- Effective contexts of use that really occur at run time.

Predictive contexts of use stand up at design time when developing the user interfaces. They act as archetypal contexts of use that define the usability domain of the user interfaces. At runtime, these predictive contexts of use may encompass a set of effective contexts of use. For example, the predictive PDA platform may fit with a Palm, a Psion or an IPAQ. Multi-targeting refers to the capacity of a user interface to withstand variations of context of use overstepping the bounds of predictive contexts of use. A multi-target user interface is able to cover at least two predictive contexts of use whatever their granularity is (e.g., workstations and PDA, or Palm and Psion, or Palm ref *x* and Palm ref *y*). This granularity depends on the level of detail addressed at design time.

To measure the extent to which a user interface is multi-target, we introduce the notion of *multi-targeting domain* (vs. *plasticity domain*).

1.3. Multi-targeting and plasticity domain

The *multi-targeting domain* of a user interface refers to the coverage of contexts of use it is able to accommodate. The subset of this domain for which usability is preserved is called *plasticity domain* (Calvary et al., 2001b). Considering that adaptation may consist in either tuning the interactive system to target the new context of use (in this case, the interactive system *ISi* switches from configuration *Cj* to configuration *Ck*. The difference between configurations may come from any software component) or switching to another interactive system *ISj*, Table 1 expresses the multi-targeting domain of any interactive system *ISi* considered in any configuration *Cj*. *ISi_j* denotes the interactive system *i* considered in configuration *j*. In this table, the contexts of use are ranked in an arbitrary way. They refer to predictive triples <user, platform, environment>.

Interactive systems / Contexts of use	C1	C2	…	Cn
IS1_C1	x	x		
IS1_C2				x
IS2_C1	x			x
IS2_C2		x		
IS2_C3				

…				
ISp_Cq		x	x	

Table 1: Multi-targeting domains. In this example, the interactive system number 1 in configuration 1 is able to cover both the contexts of use C1 and C2.

The Unifying Reference Framework provides a sound basis for identifying the multi-targeting domain of a particular user interface and for explaining how it moulds itself to target another context of use.

2. The Unifying Reference Framework

The Unifying Reference Framework covers both the design time and run time of multi-target user interfaces. As shown in Fig. 2, it is made up of three parts:

- On the left, a set of ontological models for multi-targeting that give rise to archetypal and observed models. Archetypal models are predictive models that serve as input to the design process. Observed models are effective models that guide the adaptation process at runtime.
- On the top right, a development process that explains how producing a user interface for a given archetypal context of use. This part is reasonably processed at design time.
- On the bottom right, an execution process that shows how the user interfaces and an underlying runtime infrastructure may cooperate to target another context of use.

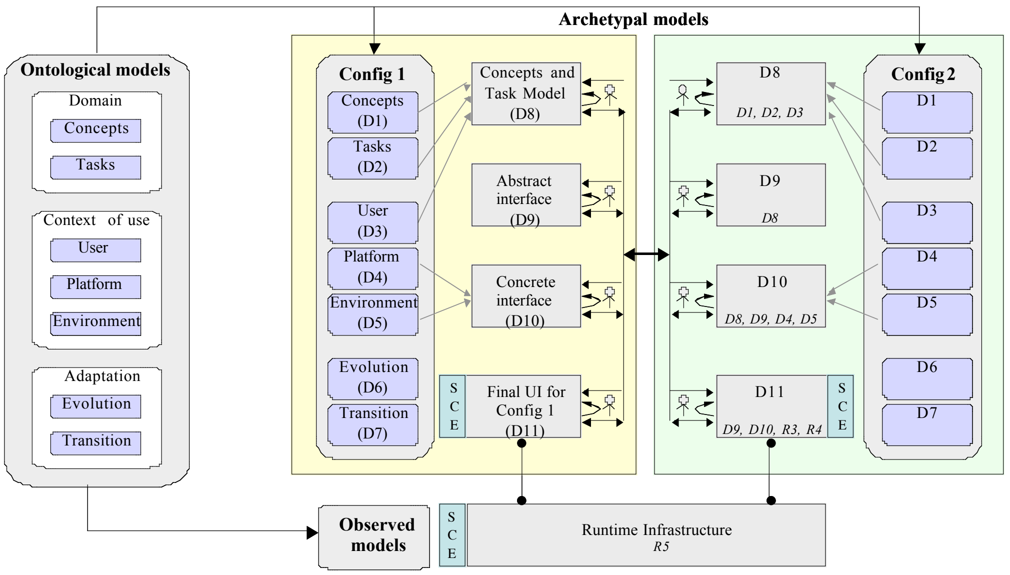


Fig. 2. The Unifying Reference Framework.

The next sections are respectively devoted to the models (ontological, archetypal and observed models), the design time and run time processes.

2.1. Ontological, archetypal and observed models for multi-targeting

"Generally, ontology is a branch of philosophy dealing with order and structure of reality. Here, we adopt Gruber's view that an ontology is an explicit specification of an abstract, simplified view of a world we desire to represent [Gruber 95]. It specifies both the concepts inherent in this view and their inter-relationships. A typical reason for constructing an ontology is to give a common language for sharing and reusing knowledge about phenomena in the world of interest" [Holsapple 02]. Here, the purpose is about multi-targeting: the reference framework identifies ontological models for multi-targeting.

Ontological models are meta-models that are independent of any domain and interactive system. Roughly speaking, they identify key-dimensions for addressing a given problem (for the purpose of this study, the multi-targeting issues). When instantiated, they give rise to *archetypal models* that, this time, are dependent of an interactive system dealing with a given domain. We distinguish three kinds of ontological models for multi-targeting:

- Domain models that support the description of the concepts and user tasks relative to a domain;
- Context models that characterize the context of use in terms of user, platform and environment;
- Adaptation models that specify both the reaction in case of change of context of use and a smoothly way for commuting.

These three kinds of ontological models remain generic. They provide tools for reasoning about multi-targeting. The *ontological domain models for multi-targeting* are based on the traditional domain models. They improve these models in order to accommodate variations of context of use. Typically, the concepts model may be an UML class diagram enriched by concepts domain variations (Thevenin, 2001). When instantiated in archetypal models, it could express the fact that a month may be modeled by an integer comprised between one and twelve. The tasks model may be a ConcurTaskTree description (Paternò, 1999) enhanced by decorations for specifying the contexts of use for which each task makes sense. Applied in archetypal models, ti could express the fact that the task "writing a paper" does not make sense on a PDA in a train.

The *ontological context of use models* provide tools for reasoning about user, platform and environment. Except the user model that comes from the traditional model-based approaches, the platform and environment models have been overlooked or ignored so far. They are now explicitly introduced to convey the physical context of use. The ontological platform model provides tool for describing the surrounding resources. For example, it may support the distinction between elementary platforms, which are built from core resources and extension resources, and clusters, which are built from elementary platforms.

An *elementary platform* is a set of physical and software resources that function together to form a working computational unit whose state can be observed and/or modified by a human user. None of these resources per se is able to provide the user with observable and/or modifiable computational function. A personal computer, a PDA, or a mobile phone, are elementary platforms. On the other hand, resources such as processors, central and secondary memories, input and output interaction devices, sensors, and software drivers, are unable, individually, to provide the user with observable and/or modifiable computational function.

Some resources are packaged together as an immutable configuration called a core configuration. For example, a laptop, which is composed of a fixed configuration of resources, is a core configuration. The resources that form a core configuration are *core resources*. Other resources such as external displays, sensors, keyboards and mice, can be bound to (and unbound from) a core configuration at will. They are *extension resources*.

A *cluster* is a composition of elementary platforms. The cluster is homogeneous when it is composed of elementary platforms of the same class. For example, DynaWall is an homogenous cluster composed of three electronic white boards. The cluster is heterogeneous when different types of platforms are combined together as in Rekimoto's augmented surfaces (Rekimoto and Saitoh, 1999).

To our knowledge, design tools for multi-platform user interfaces address elementary platforms whose configuration is known at the design stage. Clusters are not addressed. On the other hand, at the implementation level, software infrastructures such as BEACH, have been developed to support clusters built from an homogeneous set of core resources (i.e., PC's) connected to a varying number of screens (Tandler, 2001). Whereas BEACH provides the programmer with a single logical output display mapped onto multiple physical displays, MID addresses the dynamic connection of multiple input devices to a single core configuration. Similarly, Pebbles allows the dynamic connection of multiple PDA's to a single core configuration (Myers et al., 1998). None of the current infrastructures addresses the dynamic configuration of clusters, including the discovery of both input and output interaction resources. Our concept of IAM as well as the architecture developed for iRoom (Winograd, 2001) are attempts to address these issues.

From a toolkit perspective, the interactors model is one aspect of the platform model. It describes the interactors available on the platform for the presentation of the user interface. According to (Daassi, 2002), an interactor may be described by:

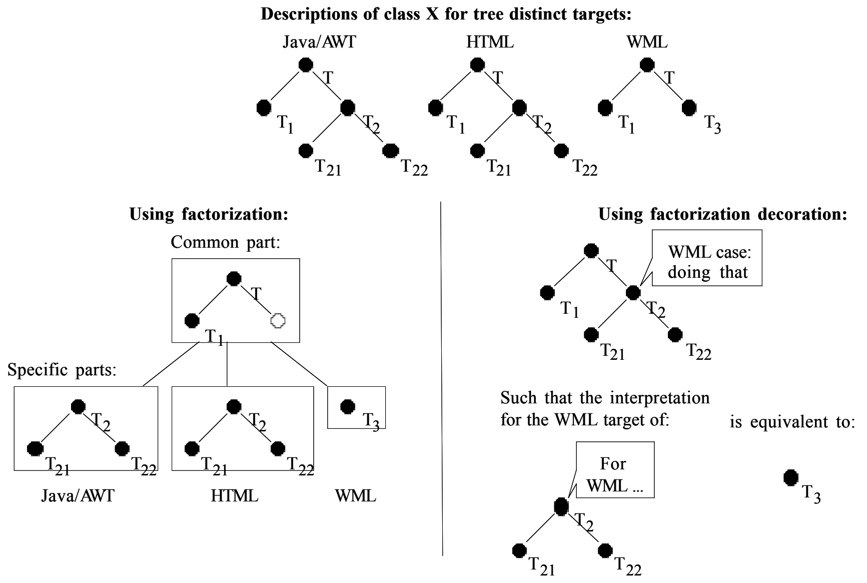
- An abstraction that models the concepts it is able to represent and the user task it supports.
- One or many presentations that each describes the rendering and manipulation of the interactor. The description is fourfold: (a) the look and feel of the interactor (b) its requirements in terms of input and output devices (for example, screen size and mouse for a graphical interactor) (c) its side effects on the context of use (for example, the noise level increasing in case of a vocal interactor) (d) the properties the presentation conveys (for example, the traditional IFIP properties (Gram and Cockton, 1996) plus the proactivity. A presentation is proactive if it guides the user in his task accomplishment).
- A control that links together the abstraction and the presentations. It qualifies the usage of this association per context of use (typical/atypical) and expresses the properties conveyed by the interactor. By opposition to the previous properties, these properties are global to the interactor. They may refer to its multi-targeting or reflexivity capacities.

The ontological environment model identifies generic dimensions for describing the surrounding environment. Salber's work is an attempt in this direction (Salber et al., 1999). The *ontological adaptation models* provide tools for describing the reaction in case of change of context of use:

- The ontological evolution model helps in specifying the interactive system or configuration of the interactive system to which the system should switch in case of changes. It supports the modeling of triggers (input in, being in or output from a context of use) and reaction;
- The ontological transition model aims at alleviating discontinuities between context changes. It offers the opportunity of specifying prologue and epilogue per change of context of use.

Whereas this set of *ontological models* is independent of any domain and context of use, the archetypal models obtained by instantiation target an interactive system in a given context of use with possible adaptations. For reasoning about multi-targeting, the unifying reference framework shown in Fig. 2 considers two sets of archetypal models: the configurations 1 and 2.

Under software engineering considerations, factorization among archetypal models may be favoured (Thevenin, 2001). *Factorization* is an operation that produces a new description composed of a description shared by targets and of descriptions specific to each target. Factorization supports multi-targeting from a "specific-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 express exceptions. Fig. 3 illustrates these two ways for sharing descriptions among archetypal models.



Descriptions of class X for tree distinct targets:
Java/AWT
HTML
WML
T
T1
T2
T21
T22
T3
Using factorization:
Common part:
Specific parts:
Using factorization decoration:
WML case: doing that
Such that the interpretation for the WML target of:
is equivalent to:
For WML ...

Fig. 3. Factorization function and factorization decorations applied to task models. At the top, three task models for Java/AWT, HTML and WML-enabled target platforms. On the bottom left, the

task model obtained from factorization. On the bottom right, the task model obtained with factorization decorations. Whereas factorization produces three specific parts linked to a shared part, the use of factorization description leads to a single shared description where exceptions are expressed as decorations.

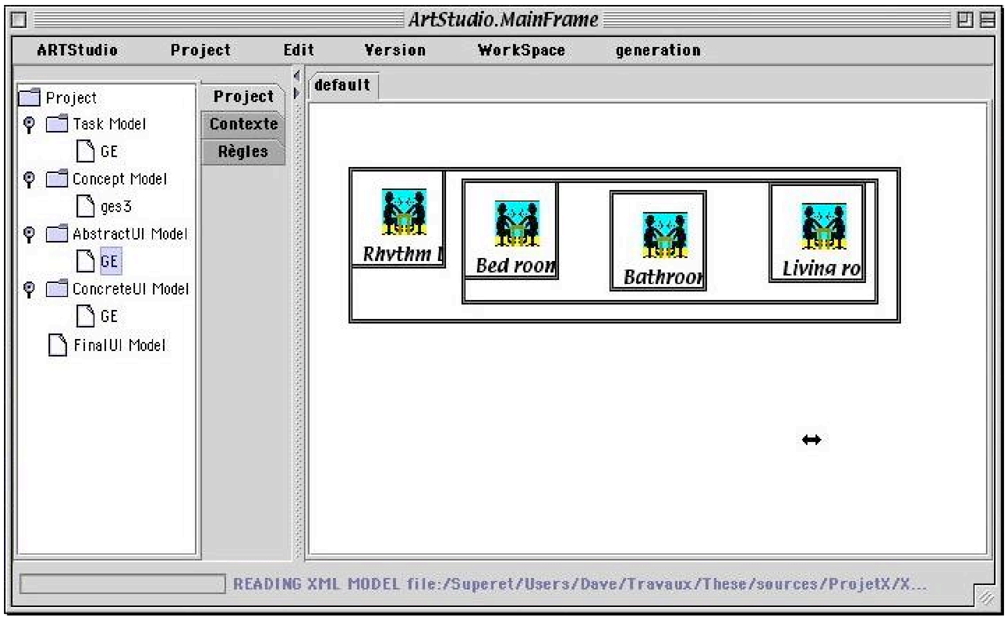
As explained in Section 1, archetypal models act as classes of potential effective models. For example, an archetypal platform PDA may fit with a Palm, a Psion or an Ipaq. The *observed models* maintain the effective models observed at run time. They may fit with one or many archetypal contexts of use in case of overlapping of multi-targeting domains. The two next sections focus on design time and run time.

2.2. Design process

Archetypal models serve as input descriptions to the development process. As they are specified manually by the developer, they are said *initial models*. The process uses a combination of *operators* to transform the initial models into *transient models* until the *final context-sensitive interactive system* is produced.

A transient model is an intermediate model, necessary for the production of the final executable user interface. Task-oriented descriptions, abstract and concrete user interfaces are typical examples of transient models:

- 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 from the interactors available on the targets. For example, in ARTStudio (Coutaz and Thevenin, 1999; Thevenin, 2001), an Abstract UI is a collection of related workspaces. The relations between the workspaces are inferred from the task relations expressed in the Concepts-and-Tasks model. Similarly, connectedness between concepts and tasks is inferred from the Concepts-and-Tasks model. Fig. 4 shows a graphical representation of workspaces in ARTstudio.
- 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 mockup than runs only within the multi-target development environment. For example, in SEGUIA (Vanderdonckt and Berquin, 1999), a Concrete UI consists of a hierarchy of Concrete Interaction Objects (CIOs) resulting from a transformation from Abstract Interaction Objects (AIOs) (Vanderdonckt and Bodart, 1993). These CIOs can be previewed in the system itself. Yet, it is not a Final UI that can be run. Fig. 5. shows a graphical representation of a Concrete UI in SEGUIA.



READING XML MODEL file:/Superet/Users/Dave/Travaux/These/sources/ProjetX/X...

Fig. 4. Workspaces in ARTStudio.

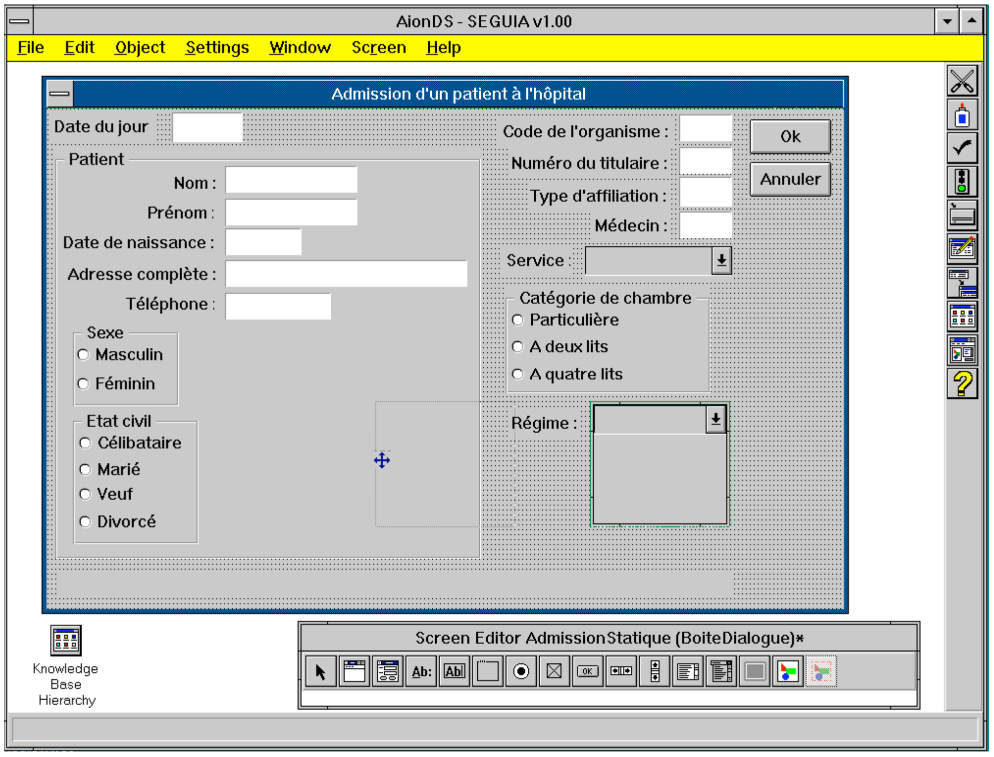


Fig. 5. Concrete UI in SEGUIA.

Transient models are produced by a combination of two kinds of operators [Calvary 02]:

- Vertical transformation that may be processed along a top-down (reification) or bottom-up (abstraction) process. Reification covers the derivation process, from top level abstract models to run time implementation. On the opposite, abstraction follows a reverse engineering process making it possible to infer abstract models from more concrete ones. This way takes into account the practice consisting in directly prototyping the concrete user interface without having produced the task-oriented specification. It supposes a total freedom in terms of entry points. An entry point marks a level of reification at which the development process (reification and/or translation) may be began. Unlike the initial process that contained an entry point only (the task oriented specification) (Calvary et al., 2001b), the unifying 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;
- Horizontal transformations, such as those performed between HTML and WML content descriptions, correspond to translations between models at the same level of reification. 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.

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 (Calvary et al., 2002).

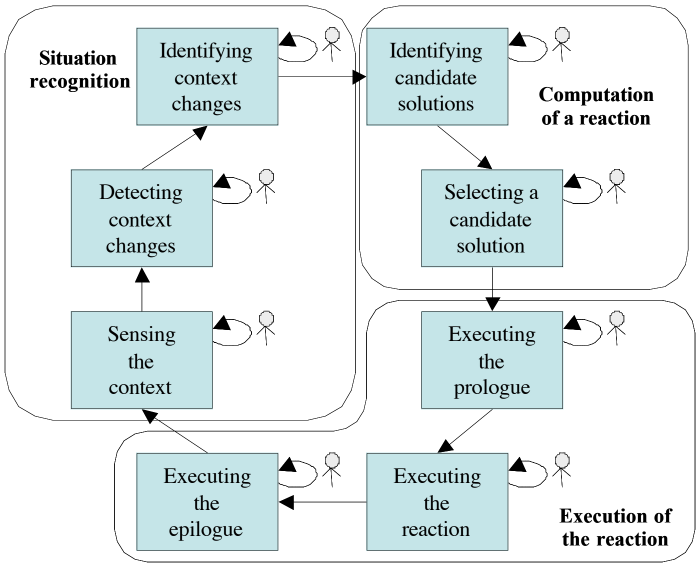
Each transformation may be performed automatically from specifications, or manually by human experts. Because automatic generation of user interfaces has not found wide acceptance in the past, the framework makes possible manual reifications and translations.

The initial models can be referenced at any stage of the reification and translation processes. In Figure 1, references are denoted by grey arrows. Delaying the reference to context models (i.e., the user, platform and environment models) at later stages, results in a larger multi-targeting domain. Whatever this reference point is, knowledge may be preserved along transformations. An italic-based notation is introduced in the unifying reference framework to make observable the life cycle of knowledge along the design time process: at level *I*, the model *Dj* is mentioned if the knowledge modeled in *Dj* is present at *I*. This traceability of knowledge may be useful for adaptation. For example, in ARTStudio (Thevenin, 2001), as the task oriented specification is already lost at the abstract user interface, the user interface is not able to accommodate 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. The next section is devoted to the run time process.

The *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 environment that supports dynamic adaptation to multiple targets.

2.3. Run-time process

As for any evolutive phenomenon, the unifying reference framework suggests to structure multi-target adaptation as a three-steps process: recognition of the situation, computation of a reaction, and execution of the reaction. Fig. 6 illustrates the run-time adaptation process.



Situation recognition
Identifying context changes
Detecting context changes
Sensing the context
Identifying candidate solutions
Computation of a reaction
Selecting a candidate solution
Executing the prologue
Executing the reaction
Executing the epilogue
Execution of the reaction

Fig. 6. Run-time adaptation process.

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., 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 suggests a third type of trigger (Schmidt, 2000) 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.

The reaction is computed in the following way: Identify candidate reactions and select one of them that best fits the situation. Reactions may be specific or may be chosen among the following generic options:

- 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 representation shown in 8a that of 8b 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 dialogue sequencing.

Some of these reactions may conserve the system state in terms of Functional Core Adaptor, Dialog Controller, etc. The persistence criteria may guide the selection among the candidate reactions. The Selection of a candidate reaction depends on migration cost. Migration cost refers to the effort the system and/or the user must put into migrating to the new UI. As discussed in (Calvary et al., 2001a), the effort is measured as a combination of criteria selected in the early phase of the development process.

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 co-operation of both. The evolution and transition models of the Unifying Reference Framework, are the location for specifying reactions that can be used by the system at run-time. 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). In practice, such system handled reactions may be performed by a cooperation of the user interface and an underlying run time infrastructure. The cooperation may be balanced in a slinky way to perform the situation recognition (S), the computation of the reaction (C) and the execution of the reaction (E). These three steps are based on archetypal and observed models.

Observed models are the effective models observed at run time. Each of the ontological models for multi-targeting may be instantiated in an effective model that describes the reality. Both these run time models *Ri* and their homologue archetypal design models *Di* may be referenced by the actor (runtime infrastructure and/or final user interface) in charge of the step of the adaptation (situation recognition, computation and execution of the reaction). As shown in Fig. 2, an italic-based notation expresses these dependences. Fig. 7 suggests a problem space for reasoning about dependences. In this figure, models are identified by a number (the same number as the one suggested on the right part of the unifying reference framework described in Figure 1). Each model *i* has two versions: a design one (*Di*) describing its archetypal dimensions; a run time version (*Ri*) maintained at run time. Any combination of models *Di* and *Rj* may contribute to the situation recognition, computation of the reaction and execution of the reaction.

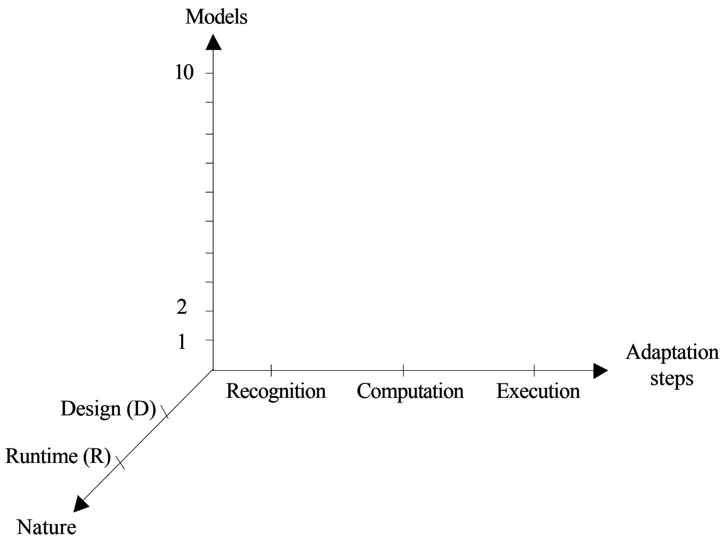


Fig. 7. Design and run time models references.

3. Applying the Unifying Reference Framework

One of the main purposes of the Unifying Reference Framework consists in its capability to express design-time and run-time consideration of various models during the development life cycle of context-sensitive applications. Once expressed in the terms of this reference framework, existing models, methods, and tools relevant to context-aware computing can be expressed and compared one with another. It is consequently possible to identify and understand the global behavior of a context-sensitive user interface by looking at how the context-sensitivity is supported.

Below we rely on this Unifying Reference Framework to express the papers presented in this Special Issue on Computer-Aided Design of User Interfaces.

Bastide *et al.* (2003) describe PetShop, a tool allowing for editing, verifying and executing the formal models (here, based on Petri nets) of highly-interactive user interfaces in the domain of safety-critical applications. An example is given about En-Route, an Air-Traffic-Control system simulator based on different models (Fig. 8):

- A *concepts model* is given that consists of three classes of domain objects along with their properties and methods: PlaneManager, Plane and Menu associated with each Plane.

- An *application model* provides access to the methods of the three classes, such as Open, Close, Remove for the Plane Class. This archetypal model is implicitly incorporated in the object architecture of the system.
- A *presentation model* specifies presentation aspects of the resulting UI in abstract terms and how to invoke them. For instance, the rendering methods specify how to graphically represent the properties of the plane and the plane itself. The activation function precises how to invoke the methods provided by the associated class in the concepts model. The rendering function is responsible for the system response when a particular method has been invoked.
- A *dialog model* contains a detailed description of the behaviours of the different classes and how to interact with them in a Petri net.

The combination of both the presentation and the dialog models forms the Abstract user interface, which is interpreted at run-time to produce the expected behaviour according to the models. PetShop does not produce per se context-sensitive user interfaces as they do not (yet) change according to any variation of user, platform (the application is itself multi-platform since implemented in Java) or environment; but it is quite conceivable that the dialog model can incorporate some context-sensitivity by supporting variations of parameters maintained in the concepts model. One of the interesting properties of Petshop is it capability to interactively modify the underlying models and to automatically reflect these changes in the interpretation of these models so as to produce the Final UI. At any time, the designer is able to modify the models, therefore supporting human intervention. This process typically represents a forward engineering approach.

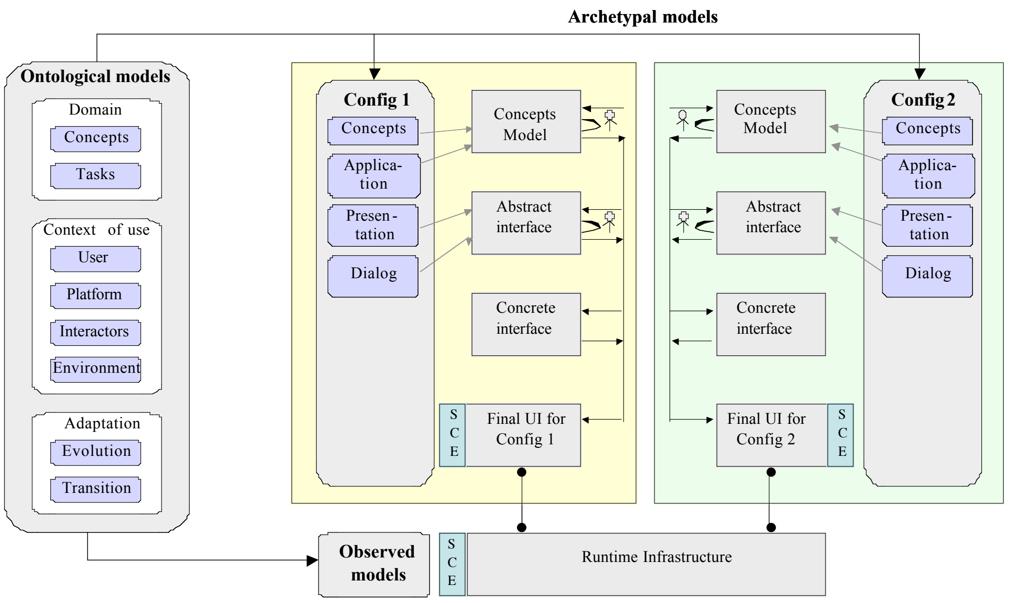


Fig. 8. The Unifying Reference Framework instantiated for PetShop.

Penner and Steinmetz (2003) follow a similar approach to PetShop by adopting a forward engineering path in te framework. The major difference is that other types of models are used at design time. The *domain model* provides the semantic basis for interaction design, allowing control system independence and an object-oriented representation of control system components. The *interaction model* provides the design knowledge required for automatic composition of appropriate tasks and abstract interactions, allowing dynamic application of the principles of usability engineering to support abstract interactions between people and systems. The *presentation model* possesses knowledge about how to select and format Concrete UIs to express interaction designs, allowing hardware and operating system independence, and a clear separation between abstract interactions (Abstract UI level) and concrete interfaces (Concrete UI level).

Luyten *et al*. (2003) present a suite of models and a mechanism to automatically produce different Final Uis at runtime for different computing platforms, possibly equipped with different input/output devices offering various modalities. This system is clearly context-sensitive since it is expressed first in a modality-independent way, and then connected to a specialisation for specific platforms. The context-sensitivity of the UI is here limited on computing platforms variations. An Abstract User Interface is maintained that contains specifications for the different rendering mechanisms (presentation aspects) and their related behaviour (dialog aspects). These specifications are written in a XML-compliant User Interface Description Language (UIDL) that is then transformed into platform-specific specifications using XSLT transformations. These specifications are then connected to a high-level description of input/output devices. A case study is presented that automatically produces three Final UIs at run-time: for HTML in a Web browser, for Java with Abstract Window Toolkit (AWT) on a Java-enabled terminal, and for Java with Swing library. Fig. 9 graphically depicts the process that is followed in this work to produce context-sensitive UIs. Note that a translation is performed at the abstract level before going down in the framework for each specific configuration (here restricted to a platform). No concepts or task models are explicitly used. The entry point of this forward engineering approach is therefore located at the level of Abstract UIs.

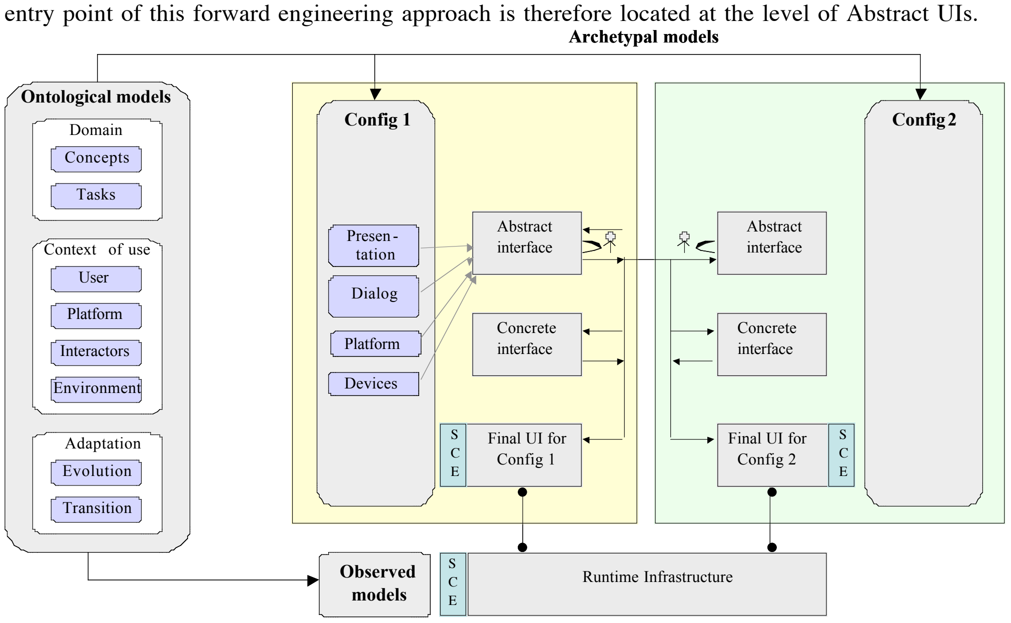


Fig. 9. The Unifying Reference Framework instantiated for SEESCOA.

Paternò & Santoro (2003) introduce a method for producing multiple Final UIs for multiple computing platforms at design time. They suggest starting with the task model of the system, then identifying the specifications of the Abstract User Interface in terms of its static structure (the presentation model) and dynamic behaviour (the dialogue model): such abstract specification will be used to drive the implementation. This time, the translation from one context of use to another is operated at the highest level: task and concepts. This allows maximal flexibility to later support multiple variations of the task depending on constraints imposed by the context of use. Here again, the context of use is limited to considering computing platforms only. The whole process is performed at design time. Consequently, there is no consideration of models at run-time. For instance, there is no embarked model that will be used during the execution of the interactive system, contrarily to the SEESCOA approach analysed above. Fig. 10 graphically depicts the Unifying Reference Framework instantiated for the method described and its underlying tool called TERESA. At the Abstract UI level, the tool provides designers with some assistance in refining the specifications for the different computing platforms considered. The Abstract UI is described in terms of Abstract Interaction Objects (AIOs) (Vanderdonckt and Bodart, 1993) that are in turn transformed into Concrete Interaction Objects (CIOs) once a specific target has been selected.

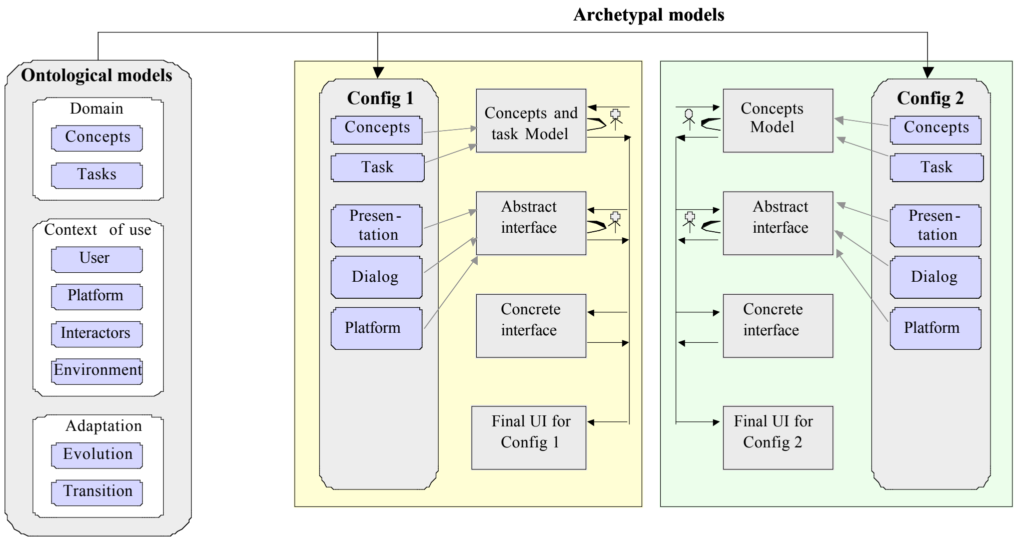


Fig. 10. The Unifying Reference Framework instantiated for TERESA.

4. Conclusion

A Unifying Reference Framework has been presented that attempts to characterize the models, the methods, and the process involved for developing user interfaces for multiple contexts of use, or so-called multi-target user interfaces. Here, a context of use is decomposed into three facets: the user, the computing platform including specific devices and the complete environment in which the user is carrying out interactive tasks with the platforms specified. Any variation of these facets can be considered as a change of the context of use that should or could be reflected in some way in the user interface.

The Unifying Reference Framework consists in an attempt to provide a better understanding and a common representation of scheme of the models, the methods and the process for multi-target user interfaces. First, it clarifies what are the models used (e.g., task, concept, presentation, dialog, user, platform, environment, interactors), when (e.g. at design-time vs. run-time), how (i.e. at the four levels of concern: task & concepts, abstract UI, concrete UI and final UI) and according to which process. Secondly, forward engineering can be expressed from the highest level of the framework (i.e. Task & concepts) to the lowest level (i.e. the Final UI), while Reverse engineering takes the inverse path. Bidirectional engineering or re-engineering (Bouillon and Vanderdonckt, 2002), that is a combination of both forward and reverse engineering, can be expressed equally. Thirdly, three types of relationships between the levels have been defined: reification for forward engineering, abstraction for reverse engineering and translation for migration from one context of use to another one. Fourthly, different entry points allow the process to be initiated at any point of the framework and not only at the top (like in a full top-down approach) or at the bottom (like in a full bottom-up approach). Fifthly, a special attention has been paid to support the expressivity of run-time mechanisms.

We hope that this Unifying Reference Framework will be used not only to characterise existing work in the domain of context-aware computing and to easily communicate this characterisation, but also to identify new opportunities for uncovered or underexplored paths in the framework. The combination of all the entry points, the models involved, the relationships used makes it very rich to express many configurations possible.

References

Bastide, R., Navarre, D., Palanque, P., 2003. A Tool-Supported Design Framework for Safety Critical Interactive Systems. Interacting with Computers, in this volume.

Benford, S., Bederson, B., Akesson, K.-P., Bayon, V., Druin, A., Hansson, P., Hourcade, J.P., Ingram, R., Neale, H., O'Malley, C., Simsarian, K.T., Stanton, D., Sundblad, Y., Taxen, G., 2000. Designing Storytelling Technologies to Encourage Collaboration Between Young Children. Proceedings of ACM Conference on Human Factors in Computer Human Interaction CHI'2000 (The Hague, 1-6 April 2000), ACM Press, New York, pp. 556–563.

Beyer, H., Holtzblatt K., 1998. Contextual Design. Morgan Kaufmann Publishers, San Francisco.

Bouillon, L., Vanderdonckt, J., 2002. Retargeting Web Pages to other Computing Platforms with VAQUITA. Proceedings of IEEE Working Conference on Reverse Engineering WCRE'2002 (Richmond, 28 October-1 November 2002), IEEE Computer Society Press, Los Alamitos, pp. 339–348.

Calvary, G., Coutaz, J., Thevenin, D., 2001a. Supporting Context Changes for Plastic User Interfaces: a Process and a Mechanism. In: Blandford, A., Vanderdonckt, J., Gray, Ph. (Eds.)Joint Proceedings of AFIHM-BCS Conference on Human-Computer Interaction

IHM-HCI'2001 (Lille, 10-14 September 2001) "People and Computers XV – Interaction without Frontiers", Vol. I, Springer-Verlag, London, pp. 349–363.

Calvary, G., Coutaz, J., Thevenin, D., 2001b. A Unifying Reference Framework for the Development of Plastic User Interfaces. In: Little, R., Nigay, L. (Eds.), Proceedings of 8th IFIP International Conference on Engineering for Human-Computer Interaction EHCI'2001 (Toronto, 11-13 May 2001), Lecture Notes in Computer Science, Vol. 2254, Springer-Verlag, Berlin, pp. 173–192.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Souchon, N., Bouillon, L., Florins, M., Vanderdonckt, J., 2002. Plasticity of User Interfaces: A Revised Reference Framework. In: Pribeanu, C., Vanderdonckt, J. (Eds.), Proceedings of 1st International Workshop on Task Models and Diagrams for User Interface Design TAMODIA'2002 (Bucharest, 18-19 July 2002), Academy of Economic Studies of Bucharest, INFOREC Printing House, Bucharest, pp. 127–134.

Card, S.K., Moran, T.P., Newell, A., 1983. The Psychology of Human-Computer Interaction. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Cockton, G., Clarke S., Gray, P., Johnson, C., 1995. Literate Development: Weaving Human Context into Design Specifications. In: Palanque, P., Benyon, D. (Eds), Critical Issues in User Interface Engineering, Springer-Verlag, London, pp. 227–248.

Coutaz, J., Rey, G., 2002. Foundations for a Theory of Contextors. In: Kolski, Ch., Vanderdonckt, J. (Eds.), Computer-Aided Design of User Interfaces III, Proceedings of 4th International Conference of Computer-Aided Design of User Interfaces CADUI'2002 (Valenciennes, 15-17 May 2002). Kluwer Academics, Dordrecht, pp. 13–33.

Crowley, J., Coutaz, J., Bérard, F., 2000. Things that See, Communications of the ACM 43 (3), 54–64.

Crowley, J., Coutaz, J., Rey, G., Reignier, P., 2002. Perceptual Components for Context-Aware Computing. Proceedings of International Conference on Ubiquitous Computing UbiComp'2002 (Göteborg, 29 September-October 1 2002). Lecture Notes in Computer Science Vol. 2498, Springer Verlag, Berlin, pp. 117–134.

Daassi, O., 2002. Les Interacteurs en Plasticité, Master of Computer Science, University Joseph-Fourier, Grenoble I.

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

Eisenstein, J., Vanderdonckt, J., Puerta, A., 2001. Model-Based User-Interface Development Techniques for Mobile Computing. In: J. Lester (Ed.), Proceedings of ACM International Conference on Intelligent User Interfaces IUI'2001 (Santa Fe, 14-17 January 2001). ACM Press, New York, pp. 69–76.

Graham, T.C.N., Watts, L., Calvary, G., Coutaz, J., Dubois, E., Nigay, L., 2000. A Dimension Space for the Design of Interactive Systems within their Physical Environments. Proceedings of ACM Symposium on Designing Interactive Systems DIS'2000 (New York, 17-19 August 2000). ACM Press, New York, pp. 406–416.

Gram, C., Cockton, G. (Eds.), 1996. Design Principles for Interactive Software. Chapman & Hall, London.

Gruber, T.R., 1995. Toward principles for the design of ontologies used for knowledge sharing. International Journal of Human-Computer Studies 43 (5/6), 907–928.

Holsapple, C.W., Joshi, K.D., 2002. A Collaborative Approach to Ontology Design. Communications of the ACM 45 (2), 42–47.

Johnson, P., Wilson, S., Markopoulos, P., Pycock, Y., 1993. ADEPT-Advanced Design Environment for Prototyping with Task Models. Proceedings of ACM Conference on

Human Aspects in Computing Systems InterCHI'93 (Amsterdam, 24-29 April 1993). ACM Press, New York, pp. 66.

Lim, K.Y., Long, J., 1994. The MUSE Method for Usability Engineering. Cambridge University Press, Cambridge.

Luyten, K., Van Laerhoven, T., Coninx, K., Van Reeth, F., 2003. Runtime transformations for modal independent user interface migration. Interacting with Computers, in this volume.

Myers, B., Stiel, H., Gargiulo, R., 1998. Collaboration using Multiple PDA's Connected to a PC. Proceedings of the ACM Conference on Computer-Supported Cooperative Work CSCW'98 (Seattle, 14-18 November 1998). ACM Press, New York, pp. 285–294.

Paternò, F., 1999, Model-based Design and Evaluation of Interactive Applications. Springer Verlag, Berlin.

Paternò, F., Santoro, C., 2003. A Unified Method for Designing Interactive Systems Adaptable To Mobile And Stationary Platforms. Interacting with Computers, in this volume.

Penner, R.R., Steinmetz, E.S., 2003. Implementation of Automated Interaction Design With Collaborative Models. Interacting with Computers, in this volume.

Rekimoto, J., Saitoh, M., 1999. Augmented Surfaces: A Spatially Continuous Workspace for Hybrid Computing Environments. Proceedings of the ACM conference on Human Factors in Computing Systems CHI'99 (Los Angeles, 1999). ACM Press, New York, pp. 378–385.

Salber, D., Dey, A.K., Abowd, G.D., 1999. The Context Toolkit: Aiding the Development of Context-Enabled Applications. Proceedings of the ACM Conference on Human Factors in Computing Systems CHI'99 (Pittsburgh, 15-20 May 15 1999). ACM Press, New York, pp. 434–441.

Schmidt, A., 2000. Implicit Human-Computer Interaction through Context. Personal Technologies 4 (2&3), 191–199.

Streitz, N., Geissler, J., Holmer, T., Konomi, S., Muller-Tomfelde, Ch., Reischl, W., Rexroth, P., Seitz, P., Steinmetz, R., 1999. I-LAND: An Interactive Landscape for Creativity and Innovation. Proceedings of the ACM Conference on Human Factors in Computing Systems CHI'99 (Pittsburgh, 15-20 May 1999). ACM Press, New York, pp. 120–127.

Tandler, P., 2001. Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices. Proceedings of International Conference on Ubiquitous Computing UbiComp'2001 (Atlanta, 30 September-2 October 2001). Springer Verlag, Berlin, pp. 96–115.

Thevenin, D., Coutaz, J., 1999. Plasticity of User Interfaces: Framework and Research Agenda. In: Sasse, A. & Johnson, Ch. (Eds.), Proceedings of IFIP Conference on Human-Computer Interaction Interact'99 (Edinburgh, 30 August-3 September 1999). IOS Press Publ., pp. 110–117.

Thevenin, D., 2001. Adaptation en Interaction Homme-Machine: Le cas de la Plasticité. Ph.D. thesis, Université Joseph Fourier, Grenoble I.

Vanderdonckt, J., Bodart, F., Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In: Ashlund, S., Mullet, K., Henderson, A., Hollnagel, E., White, T. (Eds.), Proceedings of the ACM Conference on Human Factors in Computing Systems InterCHI'93 (Amsterdam, 24-29 April 1993). ACM Press, New York, 1993, pp. 424–429.

Vanderdonckt, J., Berquin, P., 1999. Towards a Very Large Model-based Approach for User Interface Development. In: Paton, N.W., Griffiths, T. (Eds.), Proceedings of 1st International Workshop on User Interfaces to Data Intensive Systems UIDIS'99 (Edinburgh, 5-6 September 1999). IEEE Computer Society Press, Los Alamitos, pp. 76–85.

Vanderdonckt, J., Bouillon, L., Souchon, N., 2001. Flexible Reverse Engineering of Web Pages with VAQUITA. Proceedings of IEEE 8th Working Conference on Reverse Engineering WCRE'2001 (Stuttgart, 2-5 October 2001). IEEE Press, Los Alamitos, 2001, pp. 241–248.

Winograd, T., 2001. Architecture for Context. Human-Computer Interaction 16 (2-4), 401–419.