CAMELEON Project

R&D Project IST-2000-30104

Title of Document:	The CAMELEON Reference Framework
Author(s):	G. Calvary, J. Coutaz, D. Thevenin (1st and last versions of V1.1) L. Bouillon, M. Florins, Q. Limbourg, N. Souchon, J. Vanderdonckt (2nd version of V1.1) L.Marucci, F.Paternò, C.Santoro (3rd version of V1.1)
Affiliation(s):	UJF, UCL, CNUCE
Date of Document:	September 3rd, 2002
CAMELEON Document:	D1.1
Distribution:	
Keyword List:	Multi-target user interface, user interface plasticity, Cameleon reference framework
Version:	V1.1

CAMELEON Partners:

CNUCE Institute, Pisa, Italy

Université catholique de Louvain, Belgium

University of Grenoble, France

MTCI, Motorola, Turin, Italy

IS3-GICE, Paris, France

Title: CAMELEON Reference Framework	**Id Number:** D1.1

Abstract

This document covers the multi-targeting of user interfaces from the software development perspective. A taxonomic space makes explicit the core issues in multi-targeting for the development stages as well as for run-time. We propose a conceptual framework that helps structuring the development process of multi-target user interfaces as well as supporting run-time adaptation. By doing so, we are able to clearly characterize the functional coverage of current tools and identify requirements for future tools.

Title: Reference Framework	**Id Number:** D1.1

Table of Contents

1. Introduction

The field of user interface technology is expanding rapidly. This expansion is driven by three factors: technology push, users' expectations, and development costs. With the advent of digital cell phones, personal organizers, and wall-size augmented surfaces, users expect to interact with the same interactive system both *in the small* and *in the large* using multiple modalities. In addition, wireless connectivity offers new opportunities for using interactive systems in different environments (e.g., at home, in the street, at work) putting new demands on software functionalities such as context-awareness [Context 01, HUC 00, Harter 99], and dynamic discovery of interaction resources.

Meanwhile, design principles, methods and software development tools have been devised to increase systems usability while reducing development costs. But today's tools have just caught up with static workstations connected to the Ethernet. Although they have achieved a high level of sophistication, current methods and tools make a number of implicit assumptions. In particular, interactive systems are supposed to run on a personal computer located in a specific environment (typically, the workplace), exploited by users supposedly sitting in front of a screen.

In general, implicit assumptions simplify implementation issues. On the other hand, they are limiting factors for innovation. For example, windowing systems and toolkits, which set the foundations for the GUI paradigm, model windows as rectangular drawables whose borders are constrained to be parallel to that of the display. This model is based on the assumption that users keep facing the screen. With the proliferation of video beamers, it is popular to project window contents on large surfaces such as tables [Streitz 99]. In this situation, users should be able to rotate windows so that everyone can share window contents without twisting their neck. Today, rotating windows must be implemented from scratch from low level graphics primitives. Because they are screen-centric, windowing systems do not support topologies that include the surrounding environment (e.g., walls, tables, and users' location with respect to interactive surfaces, etc.). Similarly, the standard workstation is supposed to have a single mouse and keyboard. As a result, multi-user applications such as MMM [Bier 92] and Kidpad [Benford 00] whose users share the same screen with multiple mice, require the underlying toolkit and event manager to be revisited as in MID [Hourcade 99]. The UIMS technology, which inherits the assumptions of the underlying toolkits, only supports the automatic generation of WIMP user interfaces for classic personal computers and laptops.

This brief analysis shows that current software techniques primarily address the development of mono-target user interfaces. As a consequence, multi-target user interfaces are built in an ad-hoc way, resulting in high development and maintenance costs. In this document, we call for the identification of new requirements for

software tools and propose a general software framework that addresses the problem of user interface adaptation to multiple targets.

This document is structured in the following way: first, we define the notions of multi-target user interfaces as well as the challenging problem of plastic user interfaces. In Section 3, we propose the multi-context snowflake, a taxonomic space that helps clarify the software requirements for supporting the development of multi-target and plastic user interfaces. This problem space makes a clear distinction between the tools that support the design phases of the development process from those that support run-time adaptation to multiple targets. In Section 4, we propose a unifying reference framework that covers both the design time and run time of multi-target user interfaces. It structures the design phases of multi-target UI's and provides a sound basis for discussing a complementary set of tools with run time mechanisms. Section 5 is devoted to usability criteria. The definition of the terms introduced and used in this document can be found in the CAMELEON Glossary [D1.1 02].

2. Multi-context, Multi-target, Plastic User Interfaces

Multi-targeting focuses on the technical aspects of adaptation to multiple contexts. Plasticity provides a way to qualify system usability as adaptation occurs. These two concepts are presented next.

2.1. Multi-Context, Multi-target User Interfaces

A multi-context user interface is capable of supporting multiple contexts. Context is an all-embracing term. As a result, to be operational, context can only be defined in relation to a purpose, or finality. For the purpose of this study, context refers to *the context of use* of an interactive system. It is defined along three dimensions:

- 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 environment, that is the physical conditions where the interaction can take place (and/or takes place in practice).

The *user* may be modelled by a set of attributes. For example, if we refer to the Human Processor Model [Card 83], the user may be described by the attributes that characterize the perceptual, cognitive and motor abilities. In particular, disabilities may be expressed so that the designer and/or the interactive system choose the appropriate modalities. Other relevant attributes include age, motivation and experience in task accomplishment and system use.

The *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 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 exemple, the platform of the DynaWall includes three identical wall-size tactile screens mounted side by side [Streitz 99]. 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 99]. In Pebbles, PDA's can be used as input devices to control information displayed on a wall-mounted electronic board [Myers 98]. In Kidpad [Benford 00], 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. This vision has an impact on the necessity for a new underlying architecture (actually, IAM an Interaction Abstract Machine that is under development at UJF). A more formal definition of the notion of platform is provided in Section 3.

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 behaviour, either now or in the future" [Coutaz 02]. 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 concepts that are relevant to the case at hand. These include observation of users' practice [Cockton 95, Johnson 95, Lim 94, Beyer 98] 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 00]. User's location provides context for information relevance: tasks that are central in the office (e.g., writing a paper) may become secondary, or even irrelevant, in a train. For example, programming a home heating control system would make sense at home but is very unlikely on the road and would be too complex to be performed with current telephone technology.

Current research on context of use tends to gather these environment attributes into categories. Examples include physical conditions (e.g., lighting, pressure, temperature, audio, surrounding noise, time), location (e.g., absolute and relative positions, co-location), social conditions (e.g., stress, social interaction, group dynamics, collaborative tasks), and work organization (structure, user’s role).

Going one-step further than multi-targeting, we introduce the concept of plastic user interfaces.

2.2. Plastic User interfaces

The term *plasticity* is inspired from the property of materials that expand and contract under natural constraints without breaking, thus preserving continuous usage. Applied to HCI, plasticity is the "capacity of an interactive system to withstand variations of contexts of use while *preserving usability*" [Thevenin 99]. The usability of an interactive system is evaluated against a set of properties selected in the early phases of the development process. A user interface preserves usability if the properties elicited at the design stage are kept within a predefined range of values as adaptation occurs to different contexts of use. Section 5 of this document contains several usability metrics.

Although the properties developed so far in HCI [Gram 96] provide a sound basis for characterizing usability, they do not cover all aspects of plasticity. In [Calvary

01] we propose additional metrics for evaluating the plasticity of user interfaces. Whereas multi-target user interfaces guarantee technical adaptation to different contexts of use, plastic user interfaces not only guarantee technical adaptation, but guarantee usability as well. Typically, portability of Java user interfaces supports technical adaptation to different platforms but may not guarantee consistent behaviour across these platforms.

CAMELEON scenarios emphasize the need for plastic user interfaces [WP3.1 02]. Instead of forcing the user to adapt to the new context of use, the user interface adapts itself or is adapted to the new target in order to guarantee continuous usage.

2.3. Terminology: Summary

In summary, for the purpose of the CAMELEON project,

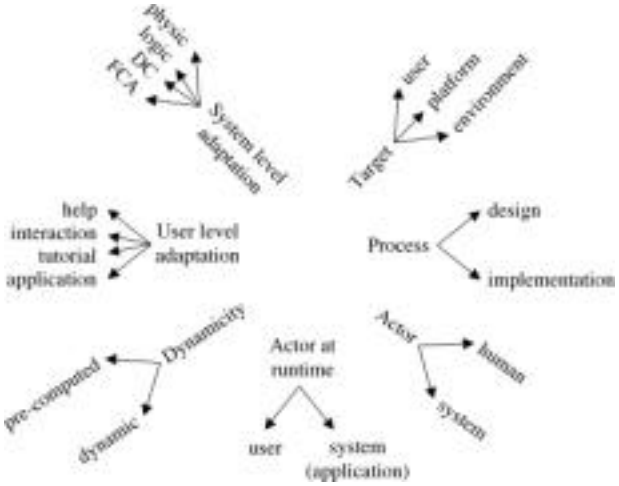
- *Context of use* is defined as the triple "user, platform, environment". It is a synonym for *target*.
- A *multi-target (or multi-context) user interface* supports multiple types of users, platforms and environments. Multi-user, multi-platform and multi-environment user interfaces are specific classes of multi-target user interfaces.
- A *multi-user interface* is a multi-target user interface that is sensitive to users variations. It is adaptable and/or adaptive to multiple archetypes (i.e., classes) of users. The environment and the platform, either are modelled as archetypes, or are implicitly represented in the system.
- A *multi-platform user interface* is a multi-target user interface that is sensitive to platforms variations. It is adaptable and/or adaptive to multiple classes of platforms. The environment and user classes, either are modelled as archetypes, or are implicitly represented in the system.
- A *multi-environment user interface* is a multi-target user interface that is sensitive to multiple environments. It is adaptable and/or adaptive to multiple classes of environments. The platform and user classes, either are modelled as archetypes, or are implicitly represented in the system. Multi-environment user interfaces are often assimilated to context-aware user interfaces [Context 01].
- A *plastic user interface* is a multi-target user interface that *preserves usability* across multiple targets. Usability is not intrinsic to a system. Usability can only be validated against a set of properties set up in the early phases of the development process.

Having defined the notions of multi-target (multi-context) and plastic user interfaces, we are now able to present a taxonomic space for multi-targeting. The goal of this taxonomy is to identify the core issues that software tools should address to support multi-targeting.

3. Taxonomic Space for Multi-target User Interfaces: a Software Tool Perspective

Figure 3.1 shows a graphic representation of the problem space for multi-targeting. Because of its shape, we call it the "*multi-target snowflake*". It can be used for characterizing existing tools or for expressing requirements for future tools. Each axis of the snowflake makes explicit a number of issues relevant to multi-targeting. These include: the types of targets that the tool supports (adaptation to platforms, environments and/or users), the stages of the development process that the tool covers (design and/or implementation), the actors that perform the adaptation of the user interface to the target (human, and/or system intervention), the dynamicity of user interfaces that the tool is able to produce (static pre-computed, and/or dynamic on the fly computed user interfaces). When considering adaptation to multiple platforms, we need to discuss the types of platforms that the tool supports as well as the way the user interface is distributed and migrates across platforms. Similar reasoning holds for characterizing different types of target environments as well as target users.

Figure 3.1 The *Multi-Target Snowflake*: a problem space for characterizing software tools, and/or for expressing requirements for software tools aimed at supporting multi-target user interfaces.



In the following sub-sections, we present each dimension of the multi-context snowflake in detail, illustrated with state-of-the-art examples. In particular, we will develop multi-platform targeting. Although as important, we are not yet in a position to provide a sound analysis for multi-user targeting. For adaptation to multi-environment, one may refer to [Coutaz 02, HUC 00, Context 01].

3.1. Target sensitivity

When considering software tools for multi-targeting, 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 types of users? Or is it a combination of platforms, environments and users?

For example, ARTStudio [Thevenin 01] addresses the problem of multi-platform targeting whereas the Context Toolkit [Dey 01] is concerned with environment sensitivity only. AVANTI, which can support visually impaired users, addresses adaptation to end-users [Stephanidis 01a]. We are not aware of any tool (or combination of tools) that supports all three dimensions of multi-targeting (i.e., users, platforms and environments).

3.2. Classes of Software tools

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

Design phases are primarily concerned with the specification tools that support modelling activities, configuration management and versioning, or code generation:

- Modelling is a foundational activity in system design. In HCI, model-based tools such as Humanoid [Szekely 96, Szekely 93], ADEPT [Johnson 93, Johnson 95] and TRIDENT [Vanderdonckt 95], have shown significant promises, not only as tools for thoughts, but as generators as well.
- Configuration management and versioning have been initiated with the emergence of large-scale software. They apply to multi-targeting as well for two reasons. First, the code that supports a particular target can result 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 each support a particular target.
- Generation has long been viewed as a reification process from high-level abstract descriptions to executable code. For the purpose of multi-targeting, we suggest generation by reification as well as by translation where transformations are applied to descriptions while preserving their level of abstraction. The Multi-target Reference Framework described in section 4 shows how to combine reification and translation.
- Abstraction results from the process of abstracting. In the context of reverse engineering, it is the opposite of reification

Implementation phases are concerned with coding. Implementation may rely on infrastructure frameworks and toolkits. Infrastructure frameworks, such as Internet or X window, 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 01]. MID is an infrastructure that extends Windows for supporting any number of mice to control a unique display [Benford 00]. UJF is currently developing IAM (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. Similar requirements motivate the blackboard-based architecture developed for iRoom [Winograd 01]. Dey et al.'s Context Toolkit, is a toolkit for developing user interfaces that are sensitive to the environment [Dey 01].

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 can be performed explicitly by humans such as system designers and implementers, and/or it can rely on dedicated tools. ARTstudio [Thevenin 02] authorizes mixed initiative cooperation.
- At the installation phase, the system may be tuned by the system manager when it is first installed. For example, a simplified version of the MacOS finder can be configured from the full set of functionalities.
- At run time, the adaptation may be performed by the user and/or the system. A UI 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.

  Adaptability and adaptativity have not yet received a common agreement in the CAMELEON consortium. For UCL and CNUCE, adaptability refers to the capacity of a UI to change its behaviour according to a small set of predefined options whereas adaptativity characterizes applications that are able to automatically modify their behaviour depending on externally-generated events.

  Actually, the right balance between adaptability and adaptivity is a tricky problem. For example, in context-aware computing, Cheverst et al. report that using context to simplify users' tasks is sometimes perceived by users as system preemption [Cheverst 01]. Adaptivity to users has been widely studied in the nineties [Browne 90] and is now available in some widely used interactive software systems (such as Microsoft Office and Amazon web site). However, there are two main risks: the tailored reaction may not match the

user's interest, due to wrong deductions, or the user may not understand the reasons for the change of the system's behaviour resulting in disorientation and frustration.

3.4. Computation of Multi-target User Interfaces

The phases that designers and developers elicit for multi-targeting have a direct impact on the types of user interfaces produced for the run time phase. Multi-target 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, implementation or installation phases of the development process: given a functional core [Arch 92], a specific user interface is generated for every known target.
- Dynamic multi-target user interfaces are computed on the fly based on run time mechanisms. Examples of run time mechanisms include the Multimodal Toolkit [Crease 00] that supports dynamic adaptation to interaction devices. FlexClock [Eisenstein 00], which dynamically adapts to window sizes, is another example.
- Hybrid multi-target user interfaces cover cases where multi-targeting is a combination of static pre-computed components with on the fly adaptation. As a general rule of thumb, pre-computation may be used for the overall structure of the user interface for satisfying response time requirements, whereas dynamic computation is applied to fine grained adjustments.

3.5. Target platforms

Experience shows that some user interfaces are aimed at simple platforms such as laptops and PDA's, while others, such as I-land [Streitz 99], require the aggregation of multiple computers. In addition, peripheral input and output devices can be connected to, or disconnected from, a computer at any time. We synthesize the variety of situations with the following notions: elementary platforms, which are built from core resources and extension resources, and clusters, which are built from elementary platforms.

3.5.1. Definitions

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, the DynaWall is an homogenous cluster composed of three electronic white boards [Streitz 99]. The cluster is heterogeneous when different types of platforms are combined together as in Rekimoto's augmented surfaces [Rekimoto 99].

3.5.2. Formal Definitions

More formally, let

- C be the set of core configurations
- E be the set of extension resources
- C', C" ≠ {}: C' C and C" = C – C'
- E', E": E' E and E" = E – E'
- $c_1, \ldots, c_n$ C , $e_1, \ldots, e_m$ E for n N*, m N
- Operational be a predicate over a set of resources that returns true when this set forms a working computational artefact whose state can be observed and/or modified by a human user.

A platform is composed of a set of core and extension resources which, connected together, form a working computational artefact whose state can be observed and/or modified by a human user:

P = { $c_1, \ldots, c_n$ } { $e_1, \ldots, e_m$ } and Operational (P)

P is an elementary platform if and only if:

¬ C', E', C", E": Operational (C' E') and Operational (C" E").

In other words, P is an elementary platform if it not possible to compose two platforms from the set of resources that constitute P.

P is a cluster if it is possible to compose two platforms from the set of resources that constitute P:

C', E', C", E": Operational (C' E') and Operational (C" E").

Note that:

- A core configuration is not necessarily operational. For example, the Intel Personal Server, a bluetooth-enabled micro-drive with no interaction device, is a core configuration but not an elementary platform: its state cannot be observed nor modified until it wirelessly connects to extension resources such as a display and/or a keyboard [Want 01].
- A laptop is an elementary platform. When augmented with extension resources such as a second mouse and sensors, it is still an elementary platform. Similarly, Rekimoto's data tiles form an elementary platform that can be dynamically extended by placing physical transparent tiles on a tray composed of an LCD flat screen display [Rekimoto 01].

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 01]. 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 98]. None of the current infrastructures addresses the dynamic configuration of clusters, including the discovery of both input and output interaction resources. The architecture developed for iRoom [Winograd 01] is an attempt to address these issues.

According to these definitions, a platform is modelled by a set of attributes that include: the type of the platform (elementary, core or extension resource, cluster), hardware features (e.g., screen resolution, processor frequency, memory size, colour availability, interaction devices) and software features (e.g., operating system, graphical toolkit).

3.6. User Interface Software Components

Multi-targeting implies that a number of software components be affected by the adaptation process. A large body of literature exists about this issue. However, because the software perspective is often mixed with the user's perception of adaptation, they do not provide a clear, non-ambiguous picture. For example, Dieterich et al. introduce five levels of adaptation: the lexical, syntactic, semantic, tasks and goals levels [Dieterich 93]. More recently, Stephanidis et al. identify the lexical, syntactic and semantic levels of adaptation using examples as definitions [Stephanidis 01b]. Typically, graphical layout reconfiguration and colour changes are defined as lexical adaptations. An example of syntactic adaptation includes switching from a "specify object first" to a "specify function first" command style.

We propose to use Arch [Arch 92], a reference software architecture model, as a sound basis for characterizing software adaptation to target changes. We then illustrate the use of Arch in relation to tasks and multi-targeting.

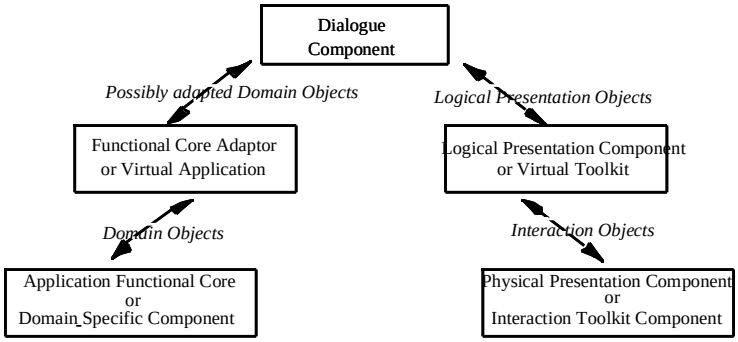
3.6.1. The Arch Model and Software Adaptation

Figure 3.2 shows the functional decomposition of an interactive system according to the Arch model [Arch 92]:

- The Functional Core (FC) implements the domain-dependent concepts and functions.
- At the other extreme, the Physical Presentation Component (PPC), which is dependent on the actual 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 physical 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 per se. The FCA can be understood as the *virtual application* layer.
- The Logical Presentation Component insulates the rendering of domain objects from the actual interaction toolkit of the target platform. It is expressed

in terms of logical presentation objects provided by a *virtual toolkit* such as Java Swing, XVT of Galaxy.

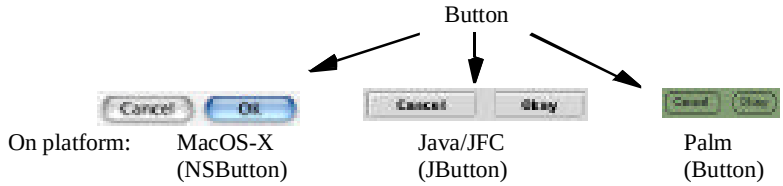
Figure 3.2 The Arch reference architecture model.



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

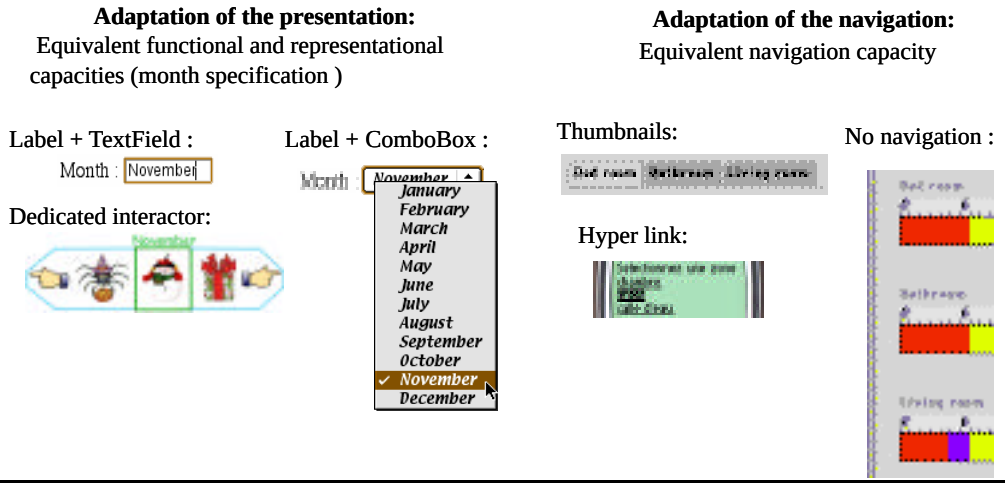
- At the Physical Presentation Component level, classes of physical interactors used for implementing the user interface are kept unchanged but their rendering and behaviour may change across platforms. For example, if a concept is rendered as a button class, this concept will be represented as a button whatever the target platform is. However, as shown in Figure 3.3, 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.

Figure 3.3. Physical level of adaptation.



- At the Logical Presentation Component level, adaptation consists of changing the representation of the domain concepts and/or of the user's tasks. For example, the selection of month can be performed through a Label+Textfield, or as a Label+Combobox, or as a dedicated physical interactor (see Figure 3.4). In a 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 93].

Figure 3.4. Logical level of adaptation. On the left, adaptation of the presentation (same task, different interactors). On the right, adaptation of the navigation (same task, different articulation).



- At the Dialog Component level, the tasks that can be executed with the system are kept unchanged but their organization is modified. As a result, the structure of the dialogue is changed. AVANTI's polymorphic tasks [Stephanidis 01a] are an example of a DC level adaptation.
- At the Functional Core Adaptor level, the natures 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 94].

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. The following section analyses the relation between tasks, multi-targeting and software adaptation at a finer grain.

3.6.2. Tasks, Multi-targeting and Software Adaptation

When multi-targeting, it is important to understand what type of tasks can actually be performed on the targets. We have identified a number of possibilities illustrated here with an interactive system that supports the visit of museum exhibits.

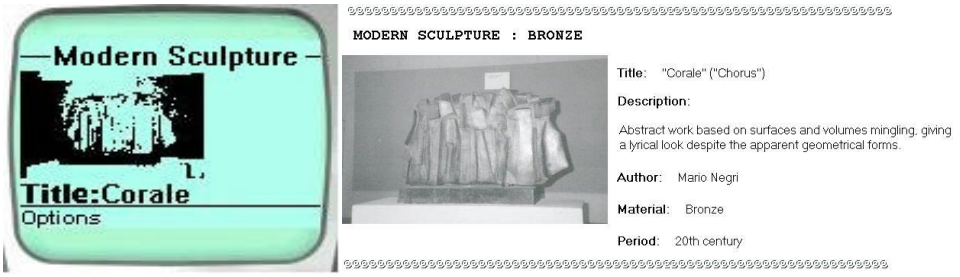
- *Same task on multiple targets with the same physical interactors*. At best, adaptation is performed at the Physical Presentation level. For the museum system, textual links are provided to access general information about the museum (how to reach, timetable, etc.) whatever the target platform is.
- *Same task on multiple targets, but with different physical interactors*. This is a Logical Presentation Adaptation. Figure 3.5 illustrates this situation. In both cases, users can select a section of the museum (e.g., Roman Archaeology, Modern Sculpture, etc). However, on the desktop system, a large, colored interactive map of the museum is at the users' disposal, whereas on the phone, because of limited screen capabilities, only a text link is available to denote a museum section.

Figure 3.5. Same task (selecting a section in a museum), different interactors: a Logical Presentation level adaptation. On the left, a map interactor shows the sections of the museum on a workstation. On the right, sections are represented as textual links on the mobile phone.

- *Same task on multiple targets, but with different sets of domain concepts*. This is a Functional Core Adaptor level adaptation. Figure 3.6 illustrates the situation for presenting information about works of art. On the desktop workstation, it is possible to access a large set of domain elements (title, image type, description, author, material, and date of creation). On the WAP phone, on the other hand, a low-resolution image of the art of work is provided (to give users a rough idea of what the work of art is about), along with the title

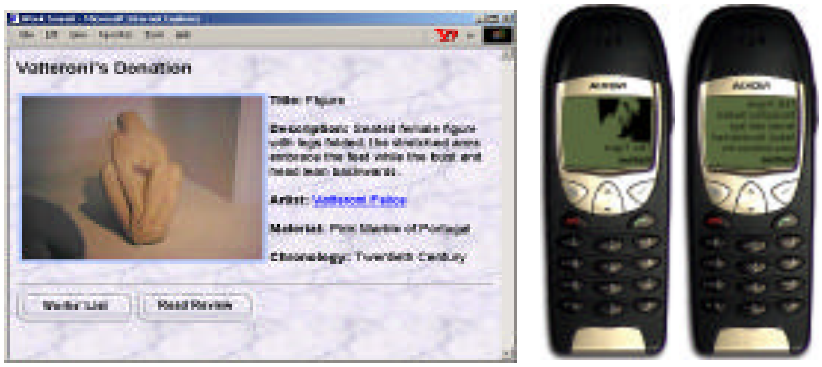
and the museum section where to find it. One will find in [D2.1 02] how the CTTE tool makes it possible to specify the domain sets for each task for distinct target platforms.

Figure 3.6. Same task (accessing a work of art), different domain concepts: a Functional Core Adaptor level adaptation. On the left, only the first-class attributes of a work of art are displayed for WAP-enabled phones. On the right, users have access to all of the attributes available about works of art.



- *Same task on multiple targets, but with different task decomposition.* This is a Dialogue Control level of adaptation. Depending on the target, the task is decomposed differently, possibly with different sets of sub-tasks. Figure 3.7 illustrates this situation for the task “access works of art”. On desktop workstations, users can accomplish additional sub-tasks, which are not supported on the WAP enabled device. For example, reading reviews of a particular work of art. This is a lengthy information-processing task that one can perform satisfactorily when sitting in front of a desktop computer, but which is unacceptable with handheld devices.

Figure 3.7. Same task (accessing a work of art), with different task decompositions: a Dialogue Component level adaptation. On the left, users using a workstation can perform subtasks such as reading reviews. These subtasks are not available on mobile phones (on the right).



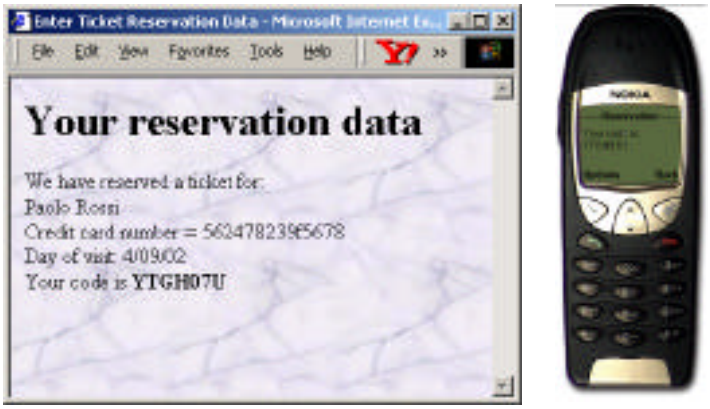
· *Same task on multiple targets, but with different temporal constraints among subtasks.* It is a Dialogue Control level adaptation. Consider the case of users who wish to electronically reserve their tickets for a particular visit in order to avoid queues. As shown in Figure 3.8, users have to provide personal information. However, while in the desktop workstation they are free to filling in the form in any order, they are constrained by the WAP interface to follow a predefined sequential order.

Figure 3.8. Same task (providing personal information), but different temporal relationships between subtasks: a Dialogue Component level adaptation. On the workstation (on the left), fields of the form can be filled in any order. On the WAP phone, a sequential order is imposed.

· *Dependencies among tasks performed on different targets*. This is a Functional Core level consideration. Consider the task for reserving tickets. With the desktop, users can access, compare and contrast the different options about the best time for visiting the museum (depending on planned exhibitions, time, etc.). Once they have selected their preferences and entered personal data, the system provides them with a special reservation code that identifies the data associated with a specific visit. It is only when users arrive at the museum that they need the reservation code. The WAP device they usually bring should therefore be able to show the code necessary to pick up the ticket at the museum and avoid queuing. Figure 3.9 illustrates the scenario. Capturing task dependencies is particularly important when the result of a available only on a specific target affects the performance of another task available on a different target. A similar situation occurs when users physically visit the museum and simultaneously annotate the most interesting works of art on their PDA. Back

at home, they would appreciate being able to receive information regarding these works of art when they first access the museum web site with their desktop workstation.

Figure 3.9. Dependencies between tasks executed on distinct target platforms. The reservation code obtained at home on the workstation is then shown on the WAP phone when needed to enter the museum.



3.7. User Interface Distribution

Clusters of platforms open the opportunity for distributing the user interface across multiple interaction devices. The granularity for distribution may vary from application level to pixel level:

- At the *application level*, the user interface is fully replicated on the platforms of the target cluster. If the cluster is heterogeneous (e.g., is comprised of a mixture of PC's and PDA's), then each platform runs a specific targeted user interface initialised, for example, from a pre-computed user interface. All of these user interfaces, however, simultaneously share the same functional core.
- At the *workspace level*, the user interface components that can migrate between platforms are workspaces. A workspace is an interaction space. It groups together a collection of interactors that support the execution of a set of logically connected tasks. In graphical user interfaces, a workspace is mapped onto the notion of window. The painter metaphor presented in Rekimoto's pick and drop [Rekimoto 97, Ayatsuka 00] is an example of distribution at the workspace level: the palettes of tools are presented on a PDA whereas the drawing area is mapped onto an electronic white board. Going one-step

further, the tools palette (possibly the drawing area) can migrate at run time between the PDA and the electronic board (See migration in section 3.8).

- At the *domain concept level*, the user interface components that can be distributed between platforms are physical interactors. Here, physical interactors allow users to manipulate domain concepts. In Rekimoto's augmented surfaces, domain concepts can be distributed between laptops and horizontal and vertical surfaces. As for Built-IT [Rauterberg 98], the topology of the rendering surfaces matters: objects are represented as 3D graphics interactors on laptops, whereas 2D rendering is used for objects placed on an horizontal surface.
- At the *pixel level*, any user interface component can be partitioned across multiple platforms. For example, in I-land, a window may lie over two contiguous white boards simultaneously. When the cluster is heterogeneous, designers need to consider multiple sources of disruption. For example, how to represent a window whose content lies across a white board and a PDA? From a user's perspective, is this desirable?

3.8. User Interface Migration

User interface migration corresponds to the transfer of the user interface between different platforms. Migration may happen at run time or between sessions only:

- *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 may be saved at multiple levels of granularity: when saved at the Dialogue Component level, the user can pursue the job 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 physical action level, that is, at the exact point within the current task. There is no discontinuity.
- *Migration between sessions* implies that the user has to quit, then restart the application from the saved state of the functional core. In this case, the interaction process is heavily interrupted. From the user's perspective, how disruption can be alleviated?

User interface migration between platforms, as well as user interface distribution put high demands on the underlying infrastructure and toolkits. They also raise interesting user-centered design issues that should be addressed within the design process. Implementation and design phases are addressed next with the presentation of the CAMELEON Reference Framework.

4. The CAMELEON Reference Framework

We first present the overall principles of the CAMELEON reference framework (Section 4.1). We then successively discuss the use of the framework for the design phase (Section 4.2) and the run-time phase (Section 4.3) of a multi-target user interface.

4.1. Principles

The CAMELEON reference framework results from two key principles:

- A model-based approach,
- Coverage of both the design and run-time phases of a multi-target user interface.

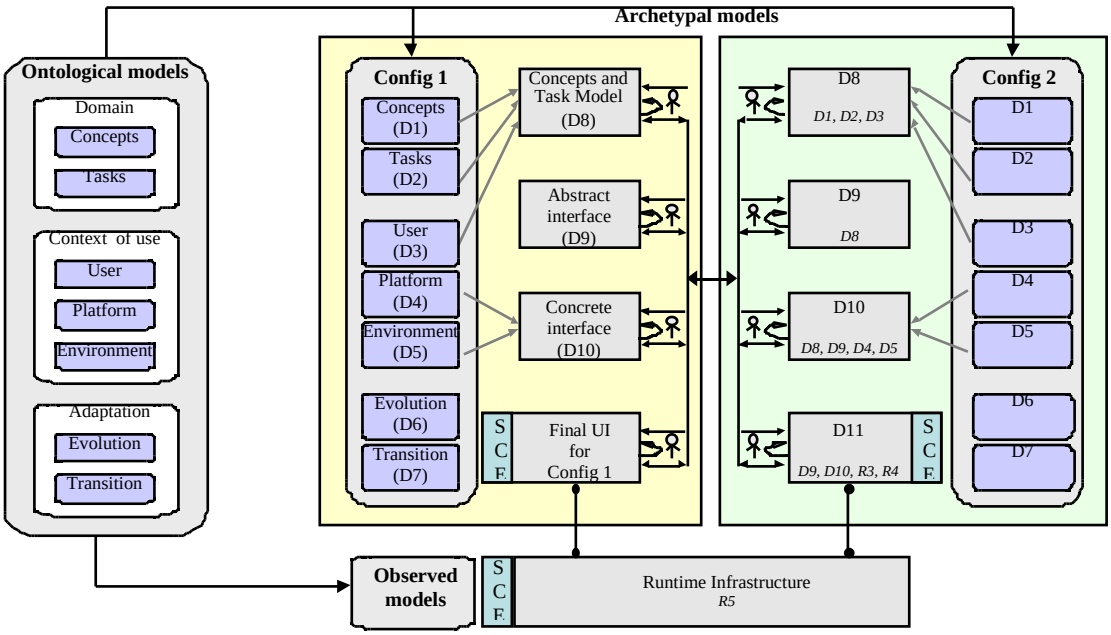
These two principles are discussed next following the overall description of the reference framework.

4.1.1. Overall Description of the CAMELEON Reference Framework

As shown in Figure 4.1, the framework is three-fold: models, design phase and run time phase.

- The *ontological models* (left side of the figure) are abstract models of the concepts (and their relationships) involved in multi-targeting. Ontological models are instantiated into archetypal and/or observed models. *Archetypal models* are declarative models that serve as input to the design of a particular interactive system. *Observed models* are executable models that support the adaptation process at run-time.
- The *design phase* (top right of the figure) complies with a structured development process whose end result is a set of executable user interfaces each targeted at particular archetypal context of use.
- For the *run-time phase* (bottom of the figure), a run-time configuration is built from a run time infrastructure and the user interfaces produced in the design phase. They, together, cooperate to support run time adaptation.

Figure 4.1. The CAMELEON Reference Framework. D_i denotes the name of a model specified above the notation (D_i). For example, D_1 denotes a Concepts model, and D_2, a Tasks model. *Italic* is used to express the presence of knowledge from an archetypal model in a model produced in the design or run-time phases. For example, the notation D_1, D_2, D_3 in the Concepts and Task Model D8 means that D8 contains knowledge about the archetypal Concepts, Task and User models of Configuration 2 (see Section 4.2.1.2). S stands for situation recognition, C for computation of a reaction, and E for execution of a reaction.



4.1.2. Model-based Approach

Model-based approaches, which rely on high-level specifications, provide the foundations for code generation and reverse engineering. As a result, they alleviate the cost of code production while improving code quality. As shown in Figure 4.2, the CAMELEON Reference Framework recommends the specification of the following classes of description:

- The *Domain Model* covers the domain concepts and users tasks. Domain concepts denote the entities that users manipulate in their tasks. Tasks refer to the activities users undertake in order to reach their goals with the system.
- The *Context of use Model* describes the context of use in terms of the user, the platform and the environment. The attributes mentioned in section 2.1 may be considered to populate this model.
- The *Adaptation Model* specifies the reaction to adopt when the context of use changes. It includes the Evolution Model that identifies the new UI to switch to, and the Transition Model that denotes the particular Transition User Interface to be used during the adaptation process. A transition UI allows the user to evaluate the evolution of the adaptation process [Barralon 02].

The models produced in the design phases (i.e., the Task-oriented specification, the Abstract UI, and the Final UI) will be presented in detail in Section 4.2. Those used in the run-time phase will be discussed in Section 4.3.

In practice, the domain model, the context of use model, and the adaptation model, which serve as the central source of knowledge in the framework, exist in three forms: ontological, instantiated, and observed. A form corresponds to a step in the life cycle of a model.

4.1.3. Life Cycle of Domain, Context of Use, and Evolution Models

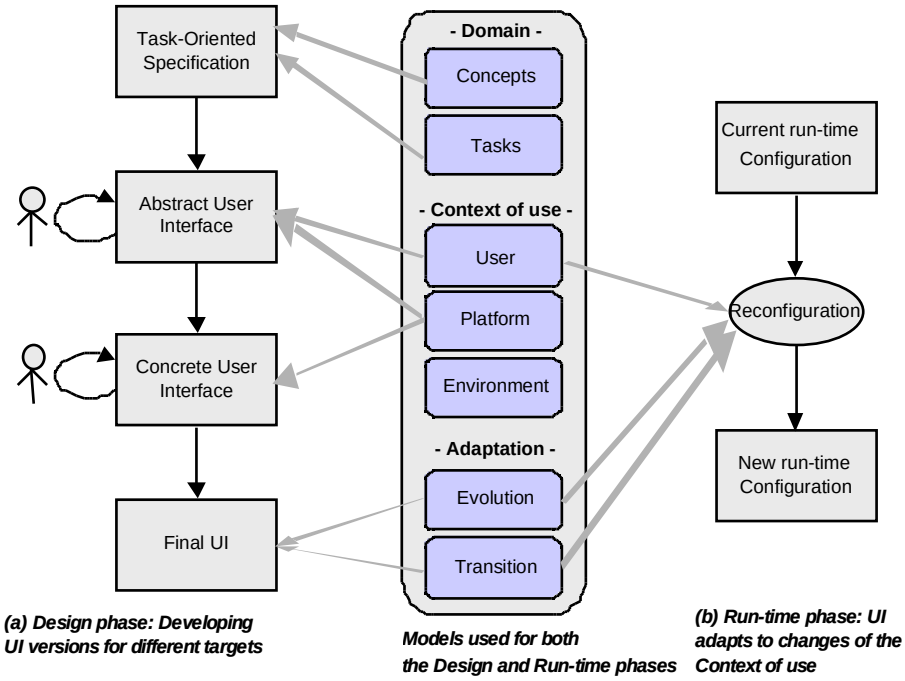
Instantiated, a model, whether it be a domain, a context of use, or an adaptation model, serves in the design phase of a user interface. *Observed*, a model, whether it be a domain, a context of use, or an adaptation model, serves at run-time for he dynamic adaptation process. *Ontological*, it serves the CAMELEON consortium as a basis for a theory.

"An ontology is a formal explicit specification of a shared conceptualisation" [Gruber 95]. "A conceptualisation … refers to an abstract model of how people think about things in the world, usually restricted to a particular subject area. An explicit specification means the concepts and the relationships of the abstract model are given explicit terms and definitions." [Gruninger 02]. For the CAMELEON consortium, the subject area is multi-targeting.

In the context of the CAMELEON reference framework, *ontological models* are meta-models that make explicit the key-dimensions for addressing multi-targeting. They are independent from any domain and interactive system, but they make explicit the CAMELEON subject area: multi-targeting. When instantiated, they give rise to *archetypal models* which, this time, depend on the domain and the interactive system being developed. As introduced above, multi-targeting involves three classes of ontological models: domain models, context of use models, and

adaptation models. All of these models are generic. They serve as tools for reasoning about multi-targeting.

Figure 4.2. Models are at the centre of the reference framework for both the design and run-time phases. In (a), domain, context of use, and adaptation models are used for statically developing different versions of a user interface each targeted at a specific target. In (b), domain, context of use, and adaptation models are used for dynamically reconfiguring a user interface in response to changes in the context of use. Arrows denote "are used by" relationships. They are shown as indicative examples of relationships. In theory, domain, context of use, and evolution models, may all impact any model developed in the design and run-time phases.



The *ontological domain models for multi-targeting* extend domain models to address multi-targeting. For example, the ontological concept model is UML classes enriched with the expression of the range of values for the domain concepts [Thevenin 99, Thevenin 01]. When instantiated as an archetypal domain model, this extension makes it possible to express that a particular interactive system handles the concept of month and that a month is an integer comprised between 1 and 12. The ontological tasks model may be ConcurTaskTrees concepts [Paternò 94] enhanced with decorations to specify the contexts of use for which each task makes sense. When instantiated as an archetypal task model,

it could express the fact that the task “reading reviews about a work of art” does not make sense on a PDA in a train.

The *ontological context of use models* supports reasoning about the target user, platform and environment. Except for the user model that comes from model-based approaches [Marucci 02], the platform and the environment models have been overlooked so far. Now a number of researchers are concerned with these notions [Dey 00] [Salber 99] [Schmidt 99]. We now explicitly introduce these models to convey the physical context of use. The ontological platform model makes explicit the dimensions for describing the software and hardware resources available. 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.

The interactors description 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 02], an (ontological) interactor comprises the following items:

- The abstraction (i.e., the data types) the interactor is able to represent as well as the user’s tasks it is able to support (e.g., selection);
- The physical presentations owned by the interactor (e.g., a graphical menu, and/or a spoken list of menu items, etc.). Each one of the possible presentations of an interactor includes: (a) the look and feel of the interactor (b) its requirements in terms of input and output resources (for example, for a graphical interactor, the screen footprint and the existence of a pointing device) (c) its side effects on the context of use (for example, for a sonic presentation, the possible increase of the noise level) (d) the properties the presentation conveys (for example, the traditional IFIP properties [Gram 96] as well as the proactive nature of the presentation. (A presentation is proactive if it guides the user in the accomplishment of the task.);
- The control that links together the abstraction and the presentations facets of the interactor. The control specifies whether the association between the abstraction and the presentations is typical or atypical for a particular context of use.

The *ontological environment model* identifies generic dimensions for describing the surrounding environment. Salber’s, Dey’s and Crowley et al.’s work are attempts in this direction [Salber 99, Dey 01, Crowley 02, Coutaz 02].

The *ontological adaptation models* make explicit the dimensions for describing the reaction in case of changes in the context of use:

- The ontological evolution model includes the notion of triggers and reaction. In turn, triggers include: entering a context of use, exiting a context of use, being in a context of use. Given a trigger, a reaction can be specified, e.g., switching to a new UI.
- The ontological transition model aims at alleviating discontinuities between changes of context. It offers the opportunity to associate a prologue and an epilogue to a reaction. These epilogue and prologue include the specification of the granularity of the state to be saved and restored along with the Transition User Interface to be executed while adaptation occurs (Cf. section 4.3).

Whereas *ontological models* are generic (i.e., they are independent from any application domain and context of use), archetypal models result from the instantiation of the ontological models for a particular application domain and a particular set of contexts of use. For example, given an interactive system to be developed, Figure 4.1 shows two sets of archetypal models denoted as Configuration 1 and Configuration 2. Configuration 1 corresponds to the set of archetypal models applicable in a context of use where as Configuration 2 corresponds to a different context of use, but both of them apply to the same interactive system.

Archetypal models act as classes of potential effective models. For example, an archetypal platform PDA may fit a Palm, a Psion or an Ipaq. The *observed models* are the effective models observed at run time. They may fit a single or multiple archetypal contexts of use in case of overlapping multi-target domains.

Archetypal and observed models respectively support the design phase and run time phase of multi-target UIs.

Having introduced the principles of the CAMELEON reference framework, we now present in detail the framework when used in the design phase of multi-targeting.

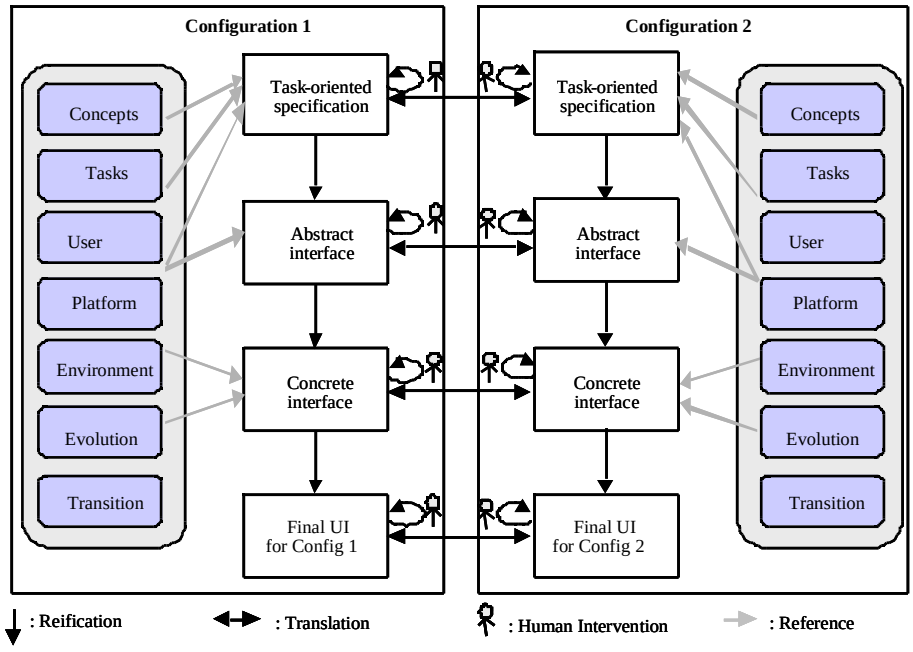
4.2. The Reference Framework for the Design Phase of Multi-targeting

The reference framework for the design phase provides designers and developers with generic principles for structuring and understanding the development process of multi-target user interfaces. It conveys a number of principles : reification, translation, human intervention, factorization, decoration, and instanciation. These are discussed next in detail followed in 4.3.2 by the description of a number of interpretations of the theoretical framework. Mathematical expressions of the operation are presented in 4.2.3.

4.2.1. Principles

The design phase is based on domain, context of use and adaptation models that are instantiations of ontological domain, context of use, adaptation models respectively. The archetypal models are referenced along the development process. As shown in Figure 4.3, the process is a combination of vertical reification and horizontal translation. A vertical reification is applied for a particular target while translation is used to express bridges between the descriptions for different targets. Reification and translation are discussed next.

Figure 4.3 The Reference Framework for the design phase of multi-target user interfaces.



4.2.1.1. Reification and Translation

Reification covers the inference process from high-level abstract descriptions to run-time code. As shown in Figure 4.3, the framework recommends 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. A 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 context of use.

- 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, 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.
- 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.
- 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 (see Section 4.3).

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 4.3, translation may 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.

By combination of reification and translation, the production of intermediary models may be avoided. As a short cut, a cross operator is introduced to both translate to another target and change the level of reification [Bouillon 02].

Reification and translation may preserve or consume knowledge. This issue is discussed in the following section.

4.2.1.2. Knowledge Life Cycle

Archetypal models can be referenced at any stage of the reification and translation process. In Figure 4.1, references to an archetypal model are denoted by grey arrows. Delaying the reference to context of use (i.e., to the user, platform and environment models) at the latest stages of the reification process, results in a wider domain for multi-targeting. Whatever this reference point is, knowledge may be preserved by the transformations.

An italic-based notation is introduced in the reference framework to make observable the life cycle of knowledge through the design phase process: at Level *I*, Model *Dj* is mentioned if knowledge modelled in *Dj* is present at *I*. Traceability of knowledge is useful for run-time adaptation. As a counter example, in ARTStudio, the task-oriented specification is lost at the Abstract User Interface level. This means that the user interface is not able to dynamically accommodate another context of use in which a different task model is required (Cf. the Museum examples in Section 3.6.2). To

overcome this drawback, first, the designer has to specify a second configuration with a different task model (as shown in Figure 4.1) leading to two pre-computed UI's, and second, the run-time infrastructure has to manage the change of context by commuting between the pre-computed user interfaces. (See Section 4.3 for the run-time phase.)

Although high-level specifications are powerful tools, they have a cost. As observed by Myers et al. about the problem of "threshold and ceiling effects" [Myers 00], high ceiling tools (i.e., powerful tools) require high learning curves. Conversely, low threshold tools, which are easy to master, do not necessarily provide the support needed. Human intervention, factorization and decoration, discussed next, are intended to alleviate this dual problem.

4.2.1.3. 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 4.3, the CAMELEON Framework for the design phase stresses human and tool cooperation in the following way: the multi-target development environment infers descriptions that the designer can then adjust to specific requirements. Decorations, presented next, is one way to perform adjustments.

4.2.1.4. Decoration

Generally speaking, a decoration is a kind of information attached to description elements. Although a decoration does not modify the description per se, it supports the expression of information that modifies the interpretation of the description.

Applied to multi-targeting, decorations are used to express exceptions to nominal conditions. Designers focus on the representative target, and then express deviations from the reference case study. We suggest three types of decorations:

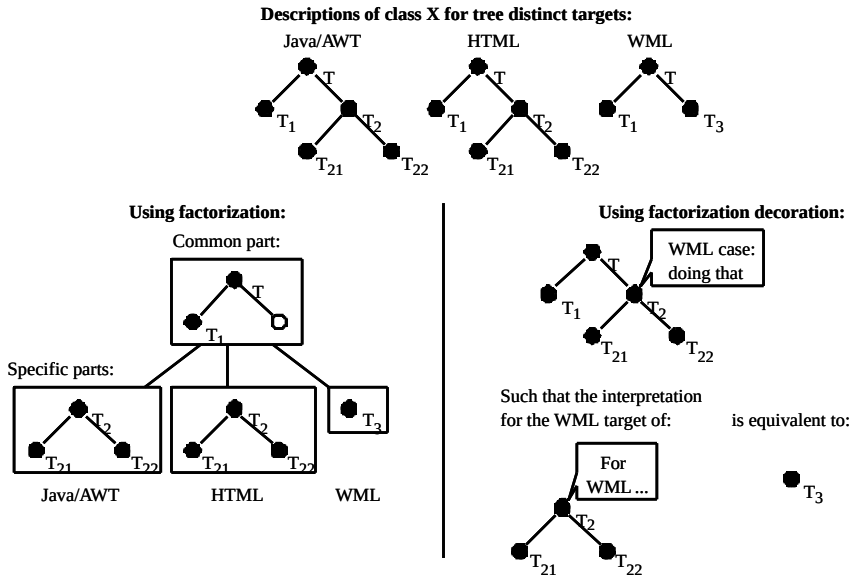
- *Directive decorations* are used when they correspond to rules that cannot be easily expressed in terms of general-purpose inference rules. For example, suppose the multi-target development environment includes the following generation rule: "any domain concept of type Integer must be represented as a Label in the Concrete UI". If the designer wants the temperature domain concept to be represented as a gauge, a directive decoration can be attached to that particular concept. Directive decorations are pro-active. Corrective decorations are reactive.

- *Corrective decorations* can be used by designers to override standard options of the multi-target UI development environment. For example, suppose that, for workstations, the multi-target UI development environment generates the Final UI without navigation (Figure 3.4). The designer can override the rule in order to use a thumbnail physical interactor, which gives access to one room at a time, by decorating the Concrete UI in order to obtain the presentation shown in Figure 3.4.
- *Factorization decorations* express exceptions to the nominal case. They are complementary to the factorization function presented next.

4.2.1.4. Factorization

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, *factorization* is an operation that produces a new description composed of a description shared by all the targets and of descriptions specific to each target. When applied to task models, factorization corresponds to AVANTI's polymorphic tasks. Figure 4.4 illustrates the principles.

Figure 4.4. 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 3 specific parts linked to a shared part, the use of factorization description leads to a single shared description where exceptions are expressed as decorations.



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.

Having presented the general principles of the CAMELEON Reference Framework for the design phase, let's see how it can be instantiated.

4.2.2. Instantiations of the Reference Framework for the Design Phase

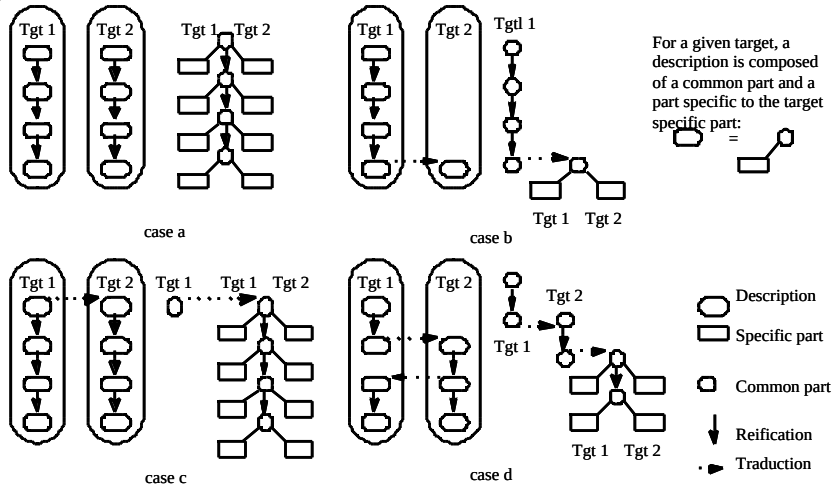
The generic representation of the Multi-target Framework shown in Figure 4.2 can be instantiated in ways that reflect distinct practices: reification and translation may be combined along different patterns; the reification process may be started from any level of abstraction; and the framework may be used to perform multi-targeting through reverse engineering as in Vaquita [Bouillon 02]. These issues are discussed next.

4.2.2.1. Combining Reification and Translation

Figure 4.5 illustrates typical patterns of reification and translation:

- In 4.5a), two final user interfaces are reified in parallel. This configuration, which depicts current practice in the absence of tool support, forces to check and maintain consistency manually between multiple versions. When factorization is applied, the consistency problem is alleviated although it is still relevant for the specific descriptions.
- Figure 4.5b) corresponds to the ideal situation: reification is applied until the very last step. Consistency maintenance is here minimal. This approach has been conducted with ARTstudio for Java-enabled target platforms. A similar approach is described in [Eisenstein 00, Eisenstein 02].
- In 4.5c), the Concepts-and-Tasks model is translated to fit another context. From there, reifications are performed in parallel. As ARTStudio does not support WML generation, this approach has been adopted to produce WAP mobile phones UIs. Sub-trees that correspond to infrequent tasks have been pruned from the original task tree developed for the Java-enabled platforms.
- Figure 4.5d) shows a mix of interleaving between reification and translation.

Figure 4.5 Different ways of combining reification and translation and their corresponding effect when factorization is performed at every stage of the reification process.



4.2.2.2. Entry Point in the Reification Process

The generic Multi-target Framework promotes a four-step process starting with domain concepts and task modelling. Although research in HCI has promoted the importance of task modelling, practitioners often skip this stage, and directly produce Concrete UI's using prototyping tools such as Flash because of the lack of tools allowing rapid prototyping from task models. This practice corresponds to the last two steps of the reification process recommended in the reference Framework. Referring to the "threshold-ceiling effect" discussed in Section 4.2.1.2, the framework can be instantiated with the number of reification steps that fits designers' culture. In other word, designers can choose the *entry point* in the reification process that best fits their practice. If necessary, the missing abstractions higher in the reification process can be retrieved through reverse-engineering [Paganelli 02].

4.2.2.3. Multi-targeting by Reverse Engineering

So far, we have presented the reference framework as a means to structure the forward development of multi-target UI's. Legacy systems, which have been designed for a particular target, must be developed from scratch to make them multi-target aware. Alternatively, they can be reverse-engineered using the CAMELEON framework where reification is complemented with an abstraction process. For

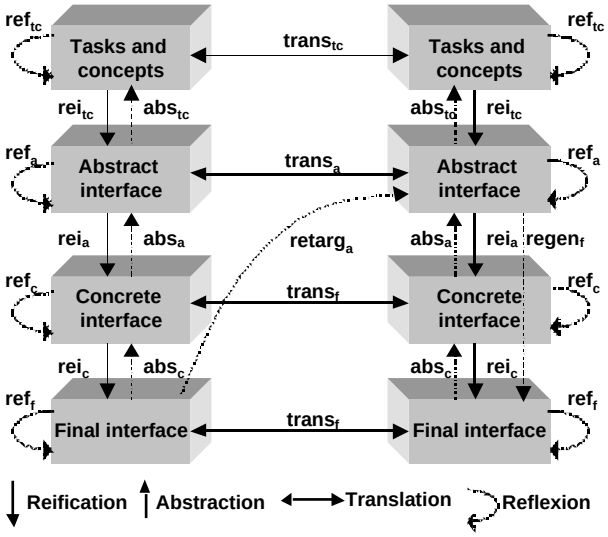
example, an Abstract UI can be inferred from a Concrete UI. In turn, a Concepts-and-Task model can be retrieved from an Abstract UI.

More generally, abstraction complements the free choice of entry points in the reification process: given a level I entry point, higher-level descriptions can be inferred through abstraction by tool support. The notion of entry point coupled with that of abstraction provides a way to address the threshold-ceiling problem.

4.2.3. Mathematical Representations of Mappings

The purpose of this section is to introduce a mathematical representation of the UI transformations performed on each level of the reference framework. Each UI transformation at any level of abstraction in Figure 4.6 can be denoted as a mathematical function.

Figure 4.6 Mathematical notations.



Let *U* be the set of all possible UIs. A *mapping M:A_B* is represented as a set

$$M = \{(x,y) \dagger x \quad A \text{ and } y \quad B\}$$

where A is called the *domain of M*, written *domain(M)* and B is called the *range of M*, written *range(M)*. A *m-to-1* mapping *M* associates one or more than one element in *domain(M)* with each element in *range(M)*. An *onto* mapping *M* associates elements of A with all elements of B. A *function f* is then defined as a *m-to-1* and *onto* mapping. If *f* is a function, then the range of *f* is also called the co-domain of *f*, written *co-domain(f).*

Reification. Reification functions are functions mapping a UI representation from one non-terminal level of abstraction to a lower level of abstraction. The functions rei_a, rei_c , rei_f respectively produce an abstract interface from a couple (task model,concepts model), a concrete interface from an abstract interface and a final interface from a concrete:

$$rei_a: TC\_AI: (t,c) \rightsquigarrow ai$$
$$rei_c: AI\_CI: ai \rightsquigarrow ci$$
$$rei_f: CI\_FI: ci \rightsquigarrow fi$$

Nota Bene: in order to avoid the multiplication of models in the definition of the origin of functions, we assume that archetypal models (e.g. a user model, a computing platform model) are given as hypotheses. A complete definition would of course include these models, but here we prefer to focus only on the origin and the destination of functions.

Abstraction. Abstraction functions are functions that map a UI representation from one non-initial level of abstraction to a higher level of abstraction. The functions abs_c, abs_a, abs_{tc} respectively produce a concrete interface from a final interface, an abstract interface from a concrete interface, and a couple (task, concepts) from an abstract interface.

$$abs_c: FI\_CI: fi \rightsquigarrow ci$$
$$abs_a: CI\_AI: ci \rightsquigarrow ai$$
$$abs_{tc}: AI\_TC: ai \rightsquigarrow (t,c)$$

Translation. Translation functions are functions that map a UI representation at a same level of abstraction, but from one context of use to another. The functions $transt_{tc}$, $trans_a$, $trans_c$, $trans_f$ respectively produce a couple (task model, concepts model), an abstract interface, a concrete interface and a final interface at the same level of abstraction for another context of use.

$$trans_{tc}: TC \text{ X } C \text{ X } C\_TC: (t_i, c_i) \text{ x } ct_i \text{ x } ct_f \rightsquigarrow (t_f,c_f)$$
$$trans_a: AI \text{ X } C \text{ X } C\_AI: ai_i \text{ x } ct_i \text{ x } ct_f \rightsquigarrow ai_f$$
$$trans_c: CI \text{ X } C \text{ X } C\_CI: ci_i \text{ x } ct_i \text{ x } ct_f \rightsquigarrow ci_f$$
$$trans_f: FI \text{ X } C \text{ X } C\_FI: fi_i \text{ x } ct_i \text{ x } ct_f \rightsquigarrow fi_f$$

where c_i, c_f denote respectively the initial and final context of use.

Reflexivity. Reflection functions are functions that map an existing UI representation at a given level of abstraction to another UI representation at the same level of abstraction for the same context of use.

$$ref_{tc}: TC \text{ X } C \_TC: (t_i,c_i) \text{ x } ct \rightsquigarrow (t_f,c_f)$$
$$ref_a: AI \text{ X } C \_AI: ai_i \text{ x } ct \rightsquigarrow ai_f$$
$$ref_c: CI \text{ X } C \_CI: ci_i \text{ x } ct \rightsquigarrow ci_f$$

$$ref_f: FI \times C\_FI: fi_i \times ct \rightsquigarrow fi_f$$

where c_i, c_f denote respectively the initial and final context of use.

Composition of functions. As it turns out to be important to perform functions one after another, to combine functions in sequence, the composition of functions needs to be defined. If *f:A_B* and *g: B_C* are two functions, then the *composition of f and g*, written as *f o g*, is defined as:

$$f \circ g = \{(x,y) \dagger \quad x \quad A, \quad y \quad C: y=g(f(x)) \}$$

that is $f \circ g = \{(x,y) \dagger \quad x \ A, \quad y \ C, z \ B$ such as $(x,z) \ B$ and $(y,z) \ C\}$. In this definition, *f* and *g* can be composed because *domain(g)=co-domain(f)*. The reverse composition *g o f* is not necessarily valid since *domain(f) ≠ co-domain(g).*

Inverse functions. Another important concept in this study of functions is inverse functions since we want to perform both forward and reverse engineering of UIs. In addition, it is important to identify when a particular UI produced by forward engineering (i.e., by applying reification, translation, reflection or any combination of them) can be completely recovered by reverse engineering (i.e., by applying abstraction, translation, reflection or any combination of them). The same question arises with respect to a UI produced by reverse engineering to be reproduced by forward engineering. These conditions can be expressed in terms of inverse functions.

Let *f* and *g* be two functions. If f(g(x)) = x and g(f(x)) = x, then *g* is the *inverse of f* and *f* is the *inverse of g.* The inverse function of a function *f* is denoted f^{-1}. In this case, *f* is said to be *inversible*. To ensure a true bidirectional engineering of UI at any level of abstraction, all involved functions should be inversible, that is the following conditions must hold:

$(t,c) \ TC, \quad a \ AI \dagger rei_a((t,c)) = a$ and $(t,c) = abs_{tc}(a)$

$a \ AI, \quad c \ CI \dagger rei_c(a) = c$ and $a = abs_a(c)$

$c \ CI, \quad f \ FI \dagger rei_f(c) = f$ and $c = abs_c(f)$

$(t_1,d_1) \ TC, \quad c_1,c_2 \ C, \quad (t_2,d_2) \ TC \dagger trans_{tc}((t_1,d_1), c_1, c_2) = (t_2,d_2)$ and $trans_{tc}((t_2,d_2), c_2, c_1) = (t_1,d_1)$

$a_1 \ AI, \quad c_1,c_2 \ C, \quad a_2 \ AI \dagger trans_a(a_1, c_1, c_2) = a_2$ and $trans_a(a_2, c_2, c_1) = a_1$

$d_1 \ CI, \quad c_1,c_2 \ C, \quad d_2 \ CI \dagger trans_c(d_1, c_1, c_2) = d_2$ and $trans_c(d_2, c_2, c_1) = d_1$

$f_1 \ FI, \quad c_1,c_2 \ C, \quad f_2 \ FI \dagger trans_f(f_1, c_1, c_2) = f_2$ and $trans_f(f_2, c_2, c_1) = f_1$

$(t_1,c_1) \ TC, \quad c \ C, \quad (t_2,c_2) \ TC \dagger ref_{tc}((t_1,c_1), c) = (t_2,c_2)$ and $ref_{tc}((t_2,c_2), c) = (t_1,c_1)$

$a_1 \ AI, \quad c \ C, \quad a_2 \ AI \dagger ref_a(a_1,c) = a_2$ and $ref_a(a_2, c) = a_1$

c_1 CI, c C, c_2 CI † $ref_c(c_1,c) = c_2$ and $ref_c(c_2, c) = c_1$
f_1 FI, c C, f_2 FI † $ref_f(f_1,c) = f_2$ and $ref_f(f_2, c) = f_1$

Restriction. For a given function to be applied in a specific computing platform, there is a need to define a condition to be satisfied when applying this function. For example, a constraint may be imposed when a translation between two computing platforms occurs, such as: "the target computing platform does not allow hierarchy of presentation elements deeper than a certain threshold". The WML language instantiates this constraint to 9 (not more than 9 presentation levels in decks and cards), while certain versions of cHTML instantiates this contraint to 4 (not more than 4 levels of cHTML tags). The restriction of function is therefore required.

A *restriction* (or *selection*) of a function *f:A_B*, denoted as $\_{cond}(f)$, is a function such that

$\_{cond}(f)$ = {(x,y) † x A, y B: y=f(x) and cond(x,y) = true} where *cond* is any first-order predicate.

Retargeting. Retargeting can now be defined as a shortcut by a composition of three functions:

$$retarg_a = \_{target}(trans_a \circ abs_c \circ abs_f)$$

where *target* is a set of first-order predicates expressing constraints applicable for a particular target. For instance, typical constraints include constraints imposed by the screen resolution, the number of colours, the availability of interaction objects, the availability of interaction resources.

Regenerating. To complete the process to obtain another UI for any other computing platform, regenerating can now be defined similarly by the composition $regen_f = rei_c \circ rei_a$ so as to represent the complete process by $regen_f \circ retarg_a$.

Idempotence. Some functions have the special property that applying them more than once to the same co-domain produces no further change after the first application. For instance, f(f(x)=f(x). To ensure a true bidirectional engineering of UI at any level of abstraction, the composition of all the functions involved and their corresponding function should be idempotent, that is the following conditions must hold:

$f = (rei_f \circ abs_c)$ f is idempotent since $fi \in FI$, i {1,…,}: $fi = f(fi) = f(f(\text{fi})) = \ldots = f\ldots_{\text{i times}}(f(\text{fi})) = f \circ f \circ_{\text{i times}} f(\text{fi})$. Similarly, $g = (rei_c \circ abs_a)$ and $h = (rei_a \circ abs_{tm})$ are idempotent.

The inversibility and idempotence of composition as introduced above guarantee a true bidirectional engineering of UI. We can observe in this case that $domain(rei_f)$

= *co-domain*(rei_f^{-1}) = *co-domain*(abs_c) and *co-domain*(rei_f) = *domain*(rei_f^{-1}) = *domain*(abs_c). If the properties of inversibility and idempotence no longer hold, then *co-domain*(abs_c^{-1}) *co-domain*(rei_f) abs_c^{-1} and rei_f are not two inverse functions of each other, but abs_c^{-1} is a restriction of rei_f . The same reasoning can be established analogously for the other levels of abstraction and for the other functions.

Having presented the CAMELEON reference framework for the design phase, we now focus on the CAMELEON reference framework for run-time.

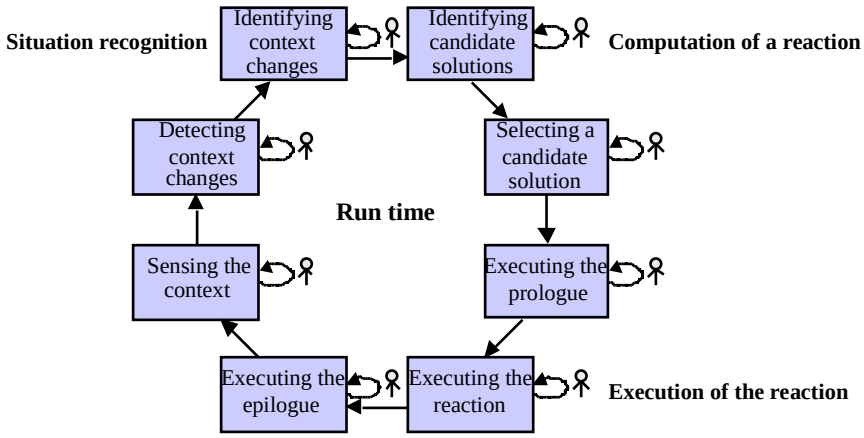
4.3. The Reference Framework for the Run-Time Phase of Multi-targeting

The CAMELEON Reference Framework for the Run-Time Phase of Multi-Targeting addresses two issues: the decomposition of the adaptation process into functional steps and the distribution of these functions across software components. These issues are presented next.

4.3.1. Run-time Adaptation: a Three-step Process

As for any evolutive phenomenon, we suggest to structure multi-target adaptation as a three-step process: recognition of the situation, computation of a reaction, and execution of the reaction. Figure 4.7 illustrates the run-time adaptation process.

Figure 4.7 Run-time adaptation process.



4.3.1.1. Situation Recognition

Recognising the situation includes the following steps:

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

In turn, the identification of context changes may trigger a reaction. There are two general types of trigger: entering a context and leaving a context. Schmidt suggests a third type of trigger [Schmidt 99] not considered in our discussion: being in a context. Triggers are combined with the AND/OR logic operators. For example, 'Leaving(C1) AND Entering(C2)' is a trigger that expresses the transition from Context C1 to Context C2. Having recognised the situation, the next step consists of computing the appropriate reaction.

4.3.1.2. Computation of a 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 to a more compact representation when the screen gets too cluttered). Adaptation may be 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 the 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 01], the effort is measured as a combination of criteria selected in the early phase of the development process.

4.3.1.3. Execution of the Reaction

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

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

Each 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 CAMELEON 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 gets tight, save the state of the running application as well as the state of the working documents (prologue), reboot the system (reaction) then re-open the working documents (epilogue).

Transition between steps means transition between states. Transition between states has been analysed since the early developments of HCI. Norman's evaluation gap, Mackinlay's *et al.* use of graphical animation for transferring cognitive load to the perceptual level [Mackinlay 91] the notion of visual discontinuity [Graham 00] etc., have all demonstrated the importance of transitions. A transition between two platforms, between executable codes, between UIs, etc. is therefore a crucial point that deserves specific research [Barralon 02]. The prologue and epilogue are here to help the designer to think about transitions.

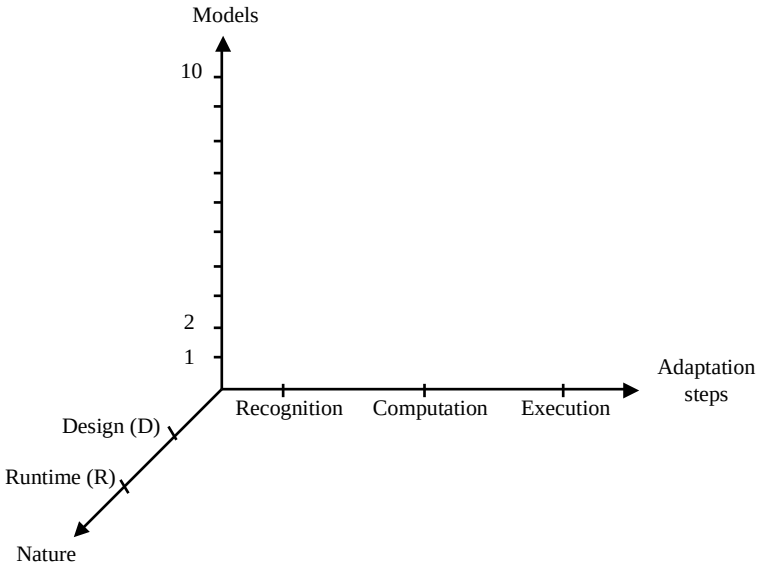
In practice, as shown in Figure 4.1, system handled reactions may be performed by a cooperation of the user interface and an underlying run time infrastructure. This slinky balance is analysed in the next section.

4.3.2. Distribution Across Components

A system-handled reaction may be performed by the cooperation of the user interface and the 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 rely on the archetypal and observed models.

Observed models are the effective models used (therefore, observed) at run time. Each one of the ontological models may be instantiated into an observed model that describes the run-time reality. The run time models *Ri* and their homologue archetypal design models *Di* may both be referenced by the actor (i.e., the run-time infrastructure and/or the final user interface) in charge of a step in the adaptation process (i.e., situation recognition, computation and execution of the reaction). Figure 4.1 shows this type of dependencies. Figure 4.8 suggests a problem space for reasoning about these dependencies. Models are identified by a number (the same number as the one suggested on the right part of the unifying reference framework of Figure 4.1). Every model *i* has two versions: a design version (*Di*) that describes the archetypal dimensions of i, and a run time version (*Ri*) maintained at run time. Any combination of the models *Di* and *Ri* may contribute to the situation recognition, the computation and execution of the reaction.

Figure 4.8. Design and run time models references.



The next section shows how the CAMELEON Reference Framework for Run-Time can be instantiated.

4.3.3. Instantiation of the Run-time Reference Framework

One contribution of the framework is to help understanding the coverage of existing tools. For example, AspectJ and the probe [Calvary 98].

4.3.3.1. Aspect-Oriented Programming and AspectJ

AspectJ provides an elegant way to transform a context non-aware final UI into a context-aware or context-sensitive final UI. Translation is performed at the lowest level of the reification process. The designer defines aspects (Aspect Oriented Programming) that will instrument the final UI code. If the aspects contain instructions for sensing the context (S), computing the reaction (C) or executing the reaction (E), then the translated UI embeds these mechanisms. In a more general way, Aspect Oriented Programming provides a way to extend pre-computed user interfaces with the expression of dynamic adaptation. The probe, which could be implemented using AOP, has the same coverage.

4.3.3.2. The probe

In the following discussion, we suppose that:

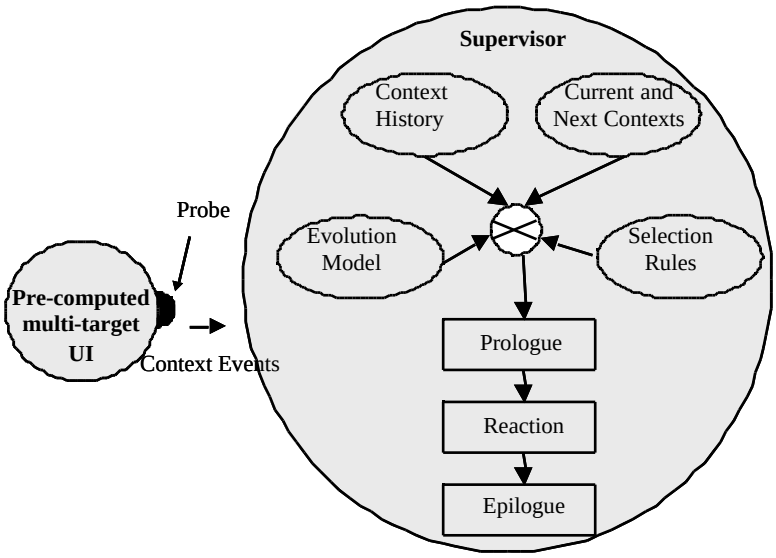
- Designers have modelled the context of use in terms of user, platform and environment.
- Context of use is sensed and modelled as software objects at the appropriate level of abstraction as in [Salber 99, Dey 01].

Figure 4.9 illustrates the principles of the probe. Write access methods to context objects are automatically augmented with spy statements that notify the context supervisor with context events. On receiving an event context, the supervisor infers the prologue, the reaction and the epilogue either:

- from the evolution model, when it exists;
- and/or from intrinsic properties of the current user interface coupled with selection rules and the history of previous contexts.

The supervisor then executes the selected prologue, the reaction and the epilogue.

Figure 4.9. The adaptation process to context changes based on a transparent probe mechanism.



With regard to the framework, the probe supports an automatic translation at the lowest level of reification. The supervisor is managed by the run-time infrastructure.

5. Usability metrics

Usability can be defined in several ways representing different design goals and requirements [Dowell 89]. In this section, usability is defined according to the human factor criteria and according to the human-computer interaction criteria.

5.1. Human Factors Criteria

Human Factor (HF) quality criteria complement Software Engineering quality factors. They meet the users perspective by defining external quality factors. The main quality criteria advanced by HF is usability. From the users' point of view, a system should be both useful and usable. Usefulness is defined as the functionalities or services that are provided to the user to accomplish his/her task. Usability refers to how these functionalities or services are provided to the user.

As argued by Olsen, programmers have probably chosen their profession because they enjoy the challenges and intricacies of the logical structures we call software [Olsen 94]. Users do rarely share this fascination. The challenge of usability is to understand users needs and apply this design knowledge to software development.

Is usability a field of science? Partly no, partly yes. On the one hand, applying design knowledge to user interfaces through guidelines or design principle is often heuristic and empirical. On the other hand, deploying systematic and reproducible knowledge to centre the development cycle on users is a scientific activity.

Usability is important to the designer because it potentially improves [Gould 85]:

- **Efficiency and Productivity.** An unusable system will distract users from their work. They will strive with the interface rather that concentrating on their task to be carried out.
- **Safety.** Usability is particularly concerned with human errors. Human error may potentially hinder safety.
- **Autonomy.** Usability potentially decreases the need for training and the learning time for a system. Support costs will be reduced accordingly.
- **Acceptance.** A usable system is more likely to be accepted by users than unusable ones.
- **Product Sales.** Usability is a major success factor for software.
- **Savings in the Development Cycle.** Adopting a usability approach from the start decreases maintenance costs.

Usability is hard to define due to the multiplicity of disciplines that contribute to the concept (e.g., psychology, human factors, software engineering). Major definitions given in the literature are surveyed.

5.1.1. The ISO 9241-11 Standard

Usability is defined by the ISO 9241-11standard as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a *specified context of use*" [ISO 91]. Effectiveness is "the accuracy and completeness with which the users achieve specified tasks". Efficiency is defined as "the resource expanded in relation to accuracy and completeness with which users achieve goals". Satisfaction is defined as "the comfort and acceptability of use by end users".

5.1.2. A Classic and General Definition

Shneiderman's [Schneiderman 98] and Nielsen's [Nielsen 94] definitions have probably been the most influential in the HCI community. Shneiderman defines usability by expressing five metrics central to achieve evaluation of human factors goal, i.e., usability. So usability is defined by Shneiderman as minimizing or maximizing these metrics:

- **Time to learn** is the time required for a typical user to learn how to achieve a particular set of tasks (and associated goals).
- **Speed of performance** is the measure of the time necessary to achieve a particular set of tasks.
- **Rate of errors by users** is a metric assessing users errors. Measuring the error rate, but also the error type, is useful to detect usability problems. This measure is particularly important for safety critical systems.
- **Retention over time** is a metric evaluating the persistence over time of the interface concepts in the user's mind. It is closely related to learning time.
- **Subjective satisfaction** is a measure that tries to grasp qualitative aspects of the interface. Subjective satisfaction is measurable through qualitative analysis (e.g., interviews facial expressions analysis).

Nielsen's usability criteria are similar to Shneiderman's but are dedicated to design analysis steps rather than on evaluation. They are: learnability, efficiency of use, memorability, few and non-catastrophic errors, subjective satisfaction.

5.2. Human computer Interaction Criteria

HCI quality criteria are decomposed in terms of properties. Dix et al. [Dix 98] and Gram et al. [Gram 96] definitions are very dialog oriented. Dix et al. top criteria for defining usability are learnability, flexibility and robustness. Gram et al. criteria are goal and task completeness, interaction flexibility and interaction robustness. Although they look similar, the contents of these criteria are not exactly the same. Furthermore, Gram et al. subtly augment these three criteria by adding internal quality criteria to the definition of usability.

Dix defines usability by raising three principles:

1. **Learnability** concerns the way a novice user can master a system. The concept of learnability is further decomposed into:
    - **Predictability**: is the support given to the user for predicting future actions based on past interaction history
    - **Synthesizebility**: is the support given to the user to assess on the effect of previous operation on current system's state.
    - **Familiarity:** is the extent to which a user's knowledge or experience of other systems or real life can be applied to the system.
    - **Generalizability:** is the support provided to the user to extend his/her knowledge of specified interaction within and across application to similar situations.
    - **Consistency**: is the likeness of behaviour arising from similar situations or similar task objectives.
2. **Flexibility** refers to the multiplicity of ways the user and the system can exchange information. Flexibility is further refined into:
    - **Dialog initiative**: refers to the ability of the user to "pre-empt" (i.e. control the initiative) the dialog with the system.
    - **Multithreading**: is the ability of the user to achieve several tasks simultaneously.
    - **Task migratability**: refers to possibility of task exchange between the system and the user. The system should be able to take in charge a task at a certain time or delegate it to the user at another time and vice versa.
    - **Substitutivity**: is the ability of a system to substitute with its current form of input output to another one.

- **Customizability**: is the UI ability to be modified by the user or by the system. If the modification is done on the users initiative the system is said adaptable. If the modification is done on the systems initiative the system is said adaptative.

3. **Robustness** concerns the system ability to support the users to achieve and assess the goal he she assigned himself herself. Robustness is further refined into:

   - **Observability**: is the ability of the user to evaluate the internal state of the system through perceivable representations. Observabilty is refined into five other principles: browsability, defaults, reachability, persistence and operation visibility. Browsability allows the users to explore systems internal state. Defaults provides user with default values as input for particular interactions with the system. Reachability allows the users to reach observable systems state. Persistence refers to the persistence of some interaction trace, operation visibility refers to the presentation to the user of the available observable states.
   - **Recoverability**: is the possibility given to the user to recover from errors.
   - **Responsiveness**: is the way the user perceives system response.
   - **Task conformance**: is the way the system supports all the tasks the user wishes to accomplish and in the way he/she understands them. Task conformance is refined into two principles: task completeness (refers to the coverage issue) and task adequacy (refers to the user's understanding).

Gram et al. refine the general software engineering criteria into HCI specific criteria. They classify usability criteria along the internal and external axis. External criteria are: goal and task completeness, interaction flexibility and interaction robustness. Internal criteria are modifiability, portability, evaluability, maintainability, run-time efficiency, integrability, functional completeness, development efficiency. The originality of Gram et al. approach lies in the fact that they explicitly link usability to internal quality factors. We consider herewith these criteria:

1. **Goal and task completeness** assert that the user should be able to reach the goals he she seeks to achieve in interaction with the system. This property is unfortunately unpredictable since the user can conceive goals that are not in the scope of the system. Task completeness appears more relevant to the developer when tasks are defined as computer support for achieving a goal.
2. **Interaction flexibility** concerns the multiple ways the user and the system may exchange information. Interaction flexibility takes into account users

differences or preferences. Interaction flexibility is further decomposed into three categories:

a) Representation of information

- **Device multiplicity** - the capacity of the system to propose multiple input/ouput devices. This property concerns the lowest level of interaction i.e., the physical level.
- **Representation multiplicity** is the flexibility in representing state elements as well as the articulation of inputs. Multiple representations concern the contents and the form of the presented information. The concept of time for instance might be represented as a digital clock, as a needle clock or as a dynamically self filling rectangle (a progression bar). The system provides multiple view of a single internal state. The system or the user decides whether a view is appropriate at a certain time during the task execution. The fact of presenting to the user multiple views at the same type is called *multimodality*. Furthermore multiple internal states of the system can be seen within a single representation. Control systems of all sorts, e.g. air traffic control, apply this principle. Multiplicity of input allows the user to communicate information to the system in various ways. For instance erasing a file can be done through direct manipulation of icons or through command line. Presenting multiple input channels simultaneously to the user is called equal *opportunity*.
- **Input Output reuse** - is the capacity of the system to use previous input or output as future input. In other words I/O reuse is the capacity of the system of recording past I/O and provide to the user to retrieve this information and reuse it for further input processes. A typical example is the cut and past function or the history recording function for command line instructions.

b) Planning of task execution

- **Human role multiplicity** refers to a cluster of user's goal adopted by a user within a particular organizational context. The concept of is used to represent collaborative aspects or at least multi-user aspects (if no collaboration exists) of interactive applications. Human role multiplicity is the capacity of the system to support several roles at the same time for multiple and single users. Database systems for instance propose different roles to the user depending on the tasks they have to perform. Roles are often:

database, designer, encoder or access forms designer. Note that the transition from one role to another refers to adaptativity.

- **Multithreading** is the capacity of the system of allowing the user to perform several tasks simultaneously
- **Non-pre-emtiveness** is the degree to which the user has control on the actions allowed to be performed. Interaction where all dialog consist in users reactions to the systems output is said system driven. On the contrary, interaction where the dialog consist in systems reactions to user input is called user driven. System driven interaction is said pre-emptive i.e., it limits the users the users freedom. User driven interaction is said non pre-emptive as it lets full initiative in the users hands. Non pre-emtiveness has a particular importance in the user's perception of flexibility.
- **Reachability** is the support given to the user to allow him to reach any system state.

c) Adaptation

- **Reconfigurability** is the ability of the user to adjust the form of input and output UNIX shells is an example of very reconfigurable interfaces as the user can define or extend commands through a powerful programming language.
- **Adaptativity** is the automatic customization of the UI by the system. Factors that are taken into account are, for instance, user experience, repetitive tasks, ambient factors, (e.g. noise, intensive light, …) , …
- **Migratability** is a property of a task that can be indifferently delegated to the system or to the user.

3. **Interaction robustness**

Interaction robustness concerns all properties that have something to do with user's failures management. An interactive system is said to be robust if the user can accomplish his/her task without irreversible mistakes and if a correct and complete picture of the task in progress is given. Robustness is refined in seven sub properties:

a) Observability is a property that states that all pertinent states (i.e., information in this case) of the system relating to the current task should be visible to the user. Due to the limited capacity of output devices all pertinent states should be at least browsable.

b) Insistence states that observability is not enough, it is sometimes necessary to attract user's attention on certain states of the system. Means

to achieve this goal are, for instance, the usage of signals of all sorts (visual, auditory) or modeless dialog boxes.

c) Honesty deals with the interpretation of signals by the user. A system may respect observability and insistence but still induce a misinterpretation of the presented system states. The honesty property states the matching of the user's representations of the UI with the actual UI. For achieving this property, the developer has to choose pertinent system states for a given task but also the right representation for a given system state. Note that flexibility has a positive impact on honesty since a flexible system adapts or is adapted to the user's characteristics.

d) Predictability is a property that allows the user to predict the future interaction with the system. Predictability is highly dependent of user s experience and knowledge but can be improved by the developer. Consistency for instance helps the user to induce knowledge on future interaction. Predictability is also concerned with temporal stability in response and execution time. A user performing similar (or perceived as) tasks expects that their respective execution time will be similar.

e) Access control is concerned with error preventing in the case of human role multiplicity. Assuming that roles are assigned to users in respect to the task they have to carry out, it is recommended to implement access control mechanism. For instance, the database administrator of a library may want to alter a record structure for the books contained in her database while a librarian is only authorised at recording loans. Access control is related to adaptativity since the system UI has to be reconfigured by the system in respect access control mechanisms.

f) Pace tolerance states an adjustment between users time and systems time.

g) Deviation tolerance states that an interactive system is should detect errors or situations where the probability of error is high. It should prevent the user from making any error and allow him her to recover from already done errors.

Gram et al. add internal quality criteria to their external quality. Internal criteria are only perceivable by the developer. As such, one could think that it does not affect the usability. One of the originality of Gram et al. work is to explicitly link internal criteria to usability (although indirectly sometimes):

1. **Modifiability** is the ease of adapting software products to change of specification. The UI part of the application is particularly concerned to extendibility.

2. **Portability** is the ease of transferring software products to various hardware and software environments.
3. **Evaluability** is the ease of a system to be evaluated on quality criteria. Evaluability of an interactive system can be achieved through logging tools recording interaction events.
4. **Maintainability** is important in interactive systems as the first cause of maintenance need is users requirement change.
5. **Run time efficiency** is the speed to which the system responds to user's input.
6. **Integrability** is the ease with which the system is integrable in existing systems.
7. **Functional completeness** states that the functional core should provide every function needed to the execution of every task. Functional completeness is the internal counterpart of task and goal completeness.
8. **Development efficiency** concerns the efficiency of the development cycle itself in terms of cost speed or other dimensions like team management and work allocation.

6. Conclusion

This paper covers multi-targeting from the software development perspective. A taxonomic space makes explicit the core issues in multi-targeting for the development stages as well as for run-time. We propose a conceptual framework that helps structuring the development process of multi-target UI's as well as supporting run-time execution. By doing so, we are able to clearly characterize the functional coverage of current tools and identify requirements for future tools.

7. References

[Arch 92] Arch, "A Metamodel for the Runtime Architecture of An Interactive System", The UIMS Developers Workshop, *SIGCHI Bulletin*, 24(1), ACM Press, 1992.

[Ayatsuka 00] Ayatsuka, Y., Matsushita, N. Rekimoto, J.: Hyperpalette: a hybrid Computing Environment for Small Computing Devices. In: CHI2000 Extended Abstracts, ACM Publ., 2000, pp. 53–53.

[Barralon 02] N. Barralon. Interfaces Homme-Machine de Transition. Master of Computer Science, University Joseph-Fourier, Grenoble I, June 2002.

[Benford 00] Benford, S. et al.: Designing Storytelling Technologies to Encourage Collaboration Between Young Children. In Proc. ACM Conference on Human Factors in Computer Human Interaction (CHI 2000), (2000), pp. 556-563.

[Beyer 98] Beyer, H., Holtzblatt K.: Contextual Design, Morgan Kaufmann Publ. (1998)

[Bier 92] Bier E., Freeman, S., Pier, K.: The Multi-Device Multi-User Multi-Editor. In Proc. of the ACM conf. On Human Factors in Computer Human Interaction (CHI92), (1992), pp. 645-646.

[Bouillon 02] Bouillon, L., Vanderdonckt, J., Souchon, N.: *Recovering Alternative Presentation Models of a Web Page with VAQUITA*, Chapter 27, in Proc. of 4th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2002 (Valenciennes, May 15-17, 2002), Kluwer Academics Pub., Dordrecht, 2002, pp. 311-322.

[Breedvelt-Schouten 97] Breedvelt-Schouten, I.M., Paterno, F.D., Severijns, C.A.: Reusable structure in task models. In: Proceedings of DSVIS'97, Design, Specification and Verification of Interactive System, Horrison, M.D., Torres, J.C. (Eds), 1997, 225–240.

[Browne 90] Browne, D., Totterdell, P., Norman, M.: Adaptive User Interface, Academic Press, Computer and People Series, 1990.

[Calvary 98] Calvary, G.: 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, 1998.

[Calvary 01] Calvary, G., Coutaz, J., Thevenin, D.: *A Unifying Reference Framework for the Development of Plastic User Interfaces*, Proceedings of 8th IFIP International Conference on Engineering for Human-Computer Interaction EHCI'2001 (Toronto, 11-13 May 2001), R. Little and L. Nigay (eds.), Lecture Notes in Computer Science, Vol. 2254, Springer-Verlag, Berlin, 2001, pp. 173-192.

[Card 83] Card S.K., Moran T.P., Newell A., *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1983.

[Cheverst 01] Cheverst, K., Davies, N., Mitchell, K., Efstratiou, C.: Using context as a Crystal Ball: Rewards and Pitfalls, *Personal and Ubiquitous Computing*, Springer Ed., 5(1), 2001, pp. 8-11.

[Cockton 95] Cockton, G., Clarke S., Gray, P., Johnson, C.: Literate Development: Weaving Human Context into Design Specifications. In: Critical Issues in User Interface Engineering, P. Palanque & D. Benyon (Eds), Springer-Verlag: London Publ., ISBN 3-540-19964-0, 1995.

[Coutaz 02] Coutaz, J., Rey, G.: Foundations for a Theory of Contextors, in Proc. CADUI'02, 2002.

[Crease 00] Crease, M., Gray, P., Brewster, S.: A Toolkit Mechanism and Context Independent Widgets, DSV-IS00, (2000), Springer Verlag, pp. 121-133.

[Crowley 00] J. Crowley, J. Coutaz, F. Bérard. Things that See, Communication of the ACM, Vol 43 (3), March 2000, pp. 54-64.

[Crowley 02] J. Crowley, J. Coutaz, G. Rey, P. Reignier. Perceptual Components for Context-Aware Computing, Ubicomp 02, Goteburg, Suède, September 2002.

[D1.1 02] G. Calvary, J. Coutaz, D. Thevenin, L. Bouillon, M. Florins, Q. Limbourg, N. Souchon, J. Vanderdonckt, L. Marucci, F. Paternò, C. Santoro, The CAMELEON Glossary, Deliverable D1.1 Companion, CAMELEON project, 2002.

[D2.1 02] G. Mori, L. Paganelli, F. Paternò, C. Santoro, Tools supporting transformations for multi-platform applications, Deliverable D2.1, CAMELEON project, 2002.

[Daassi 02] Daassi, O. Les Interacteurs en Plasticité, Master of Computer Science, University Joseph-Fourier, Grenoble I, June 2002.

[Dey 00] Dey, A.K, Abowd, G.D Towards a Better Understanding of Context and Context-Awareness, CHI 2000 Workshop on The What, Who, Where, When, and How of Context-Awareness, The Hague, Netherlands, April 1-6 2000.

[Dey 01] Dey, A., Abowd, G., Salber, D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Human Computer Interaction, Lawrence Erlbaum Publ., 16(2-4), (2001), pp. 97-166.

[Dieterich 93] Dieterich et al.: State of the Art in Adaptive user Interfaces. In Adaptive User Interfaces, Elsevier Publ., Human Factors in Information Technology series, 1993.

[Dix 98] Dix, A., Finalay, J., Abowd, G. and Beale, R. *Human-Computer Interaction*, Prentice Hall Europe, London, 2nd Edition, 1998.

[Dowell 89]Dowell, J., Long, J.: Toward a conception for an engineering discipline of human factors, Ergonomics, Vol. 32 (11), 1989, pp. 1513–1535.

[Eisenstein 00] Eisenstein J., Vanderdonckt, J. Puerta, A.: Adapting to Mobile Contexts with User-Interfaec Modeling. In: Proc. of 3rd IEEE Workshop on Mobile Computing Systems and Applications WMCSA 2000 (Monterey, December 7-8, 2000), IEEE Press, Los Alamitos, 2000.

[Eisenstein 02] Eisenstein, J., Rich, C.: *Agents and GUIs from Task Models*, in Proc. of ACM Conf. on Intelligent User Interfaces IUI'2002 (San Francisco, Junary 13-16, 2002), ACM Press, New York, 2002, accessible at http://www-scf.usc.edu/ ~jeisenst/papers/iui02.pdf

[Gould 85] J.D. Gould and C.Lewis, Design for usability: Key principles and what designers think, *Communication of the ACM*, 28(3), 300-311, 1985.

[Graham 00] Graham, T.C. N., Watts, L., Calvary, G., Coutaz, J., Dubois, E., Nigay, L. A Dimension Space for the Design of Interactive Systems within their Physical Environments, DIS2000, ACM Publ. New York, 2000, pp. 406–416.

[Gram 96] Gram, C., Cockton, G. Ed.: Design Principles for Interactive Software. Chapman & Hall, 1996.

[Gruber 95] Gruber, T.R. Toward principles for the design of ontologies used for knowledge sharing. Int. J. Hum. Comput. Stud. 43, 5/6, 1995, pp. 907-928.

[Harter 99] Harter, A., Hopper, A., Steggles, P., Ward, A., and Webster, P.: The anatomy of a context-aware application (p. 59-68), Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking , August 15 - 19, 1999, Seattle, WA USA.

[Gruninger 02] M. Gruninger, J. Lee. Ontology: Application and Design, Communications of the ACM, February 2002, Vol. 45, N°2, pp. 39-41.

[Hourcade 99] Hourcade, J., Bederson, B.: Architecture and Implementation of a Java Package for Multiple Input Devices (MID), CS-TR-4018, UMIACS-TR-99-26, May 1999.

[HUC 00] HUC 2K, First workshop on Resource Sensitive Mobile Human-Computer Interaction, 2000.

[Gram 96] Design Principles for Interactive Software, Gram, C. and Cockton, G. (eds), Chapman & Hall, 1996.

[ISO 91] ISO9241-11 Ergonomic requirement for office works with VDT's – guidance on usability. Technical report, International Standard Organisation, 1991.

[Johnson 93] Johnson, P. Wilson, S., Markopoulos, P., Pycock, Y.: ADEPT-Advanced Design Environment for Prototyping with Task Models. In: InterCHI'93 proceedings, 1993, pp. 66.

[Johnson 95] Johnson, P. Johnson, H. Wilson, S.: Rapid Prototyping of User Intefaces Driven by Task Models, Scenario-based design: envisioning work and technology in system development, J. Carroll (ed.), John Wiley&Sons, 1995.

[Lim 94] Lim, K. Y., Long, J.: The MUSE Method for Usability Engineering, Cambridge Univ. Press, 1994.

[Marucci 02] Marucci L, Paternò F. Supporting adaptivity with heterogeneous platforms through user models, in Proc. Mobile HCI 2002, Springer Verlag publ., 2002.

[Mackinlay 91] Mackinlay, J.D., Robertson, G.G, Card, S.K.: *The Perspective Wall: Detail and Context Smoothly Integrated*, in Proc. of ACM Conference on Human Factors in Computing Systems CHI'91 (New Orleans, April 27 - May 2, 1991), ACM Press, New York, 1991, pp. 173-179.

[Myers 00] Myers, B., Hudson, S., Pausch, R.: Past, Present, Future of User Interface Tools. Transactions on Computer-Human Interaction, ACM, 7(1), March 2000, pp. 3-28.

[Myers 98] Myers, B., et al.: Collaboration using multiple PDA's connected to a PC. In Proc. Of the ACM Conf. On CSCW (CSCW98), 1998, pp. 285-294.

[Nielsen 94] Nielsen, J. *Usability Engineering,* Morgan Kauffman, San Francisco, 1994.

[Olsen 94] Olsen, D.R. *Developing User Interfaces,* Morgan Kauffman, San Francisco, 1994.

[Paganelli 02] Paganelli L., Paternò F., Automatic Reconstruction of the Underlying Interaction Design of Web Applications, Proceedings SEKE 2002, ACM Press

[Paternò 94] Paternò, F., Leonardi, A. *A Semantics-based Approach to the Design and Implementation of Interaction Objects*, Computer Graphics Forum, Blackwell Publisher, Vol.13, N.3, pp.195-204, 1994.

[Paternò 02] Paternò, F., Santoro C., One Model, Many Interfaces, in Proc. CADUI 2002, Kluwer.

[Rauterberg 98] Rauterberg, M. et al.: BUILT-IT: A Planning Tool for Consruction and Design. In Proc. Of the ACM Conf. In Human Factors in Computing Systems (CHI98) Conference Companion, 1998, pp. 177-178.

[Rekimoto 99] Rekimoto, J., Saitoh, M. Augmented Surfaces: A spatially continuous workspace for hybrid computing environments. In Proc. of the ACM conference on Human Factors in Computing Systems (CHI99), ACM Press, 1999, pp. 378-385.

[Rekimoto 01] Rekimoto, J., Ullmer, B., Oba, H. DataTiles: A modular Platform for Mixed Physical and Graphical Interactions. In Proc. Of the ACM conference on Human Factors in Computing Systems (CHI2001), ACM Press, 2001, pp. 269-276.

[Rekimoto 97] Rekimoto, J. Pick and Drop: A Direct Manipulation Technique for Multiple Computer Environments. In Proc. Of UIST97, ACM Press, 1997, pp. 31-39.

[Robertson 91] Robertson, G., Mackinlay, J., Card, S.: Cone Trees: Animated 3D Visualizations of Hierarchical Information. In: Proc. CHI90, ACM Publ., 1991, pp. 189-194.

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

[Schmidt 99] Schmidt, A.: *Implicit human-computer interaction through context,* Proc. of 2nd Workshop on Human Computer Interaction with Mobile Devices MobileHCI'0, Edinburgh, August 1999.

[Schneiderman 98] Schneiderman, S. *Designing the User Interface. Strategies for Effective Human-Computer Interaction.* Addisson-Wessley, Reading (MA), third edition, 1998.

[Context 01] Special Issue on Context-Aware Computing, Human Computer Interaction, Lawrence Erlbaum, 16(2-4), 2001.

[Stephanidis 01a] Stephanidis, C. et al.: A case study in Unified User Interface Development: The AVANTI Web Browser. User Interface for All, Concepts, methods and tools, Lawrence Erlbaum, Publ., ISBN 0-8058-2967-9, 2001, pp. 525-568.

[Stephanidis 01b]Stephanidis, C., Savidis, A.: Universal Access in the Information Society: Methods, Tools, and Interaction Technologies. Journal of the Universal Access in Information Society (UAIS), 1(1), June 2001, Springer Publ., pp. 40-55.

[Streitz 99] Streitz, N. et al., I-LAND: An interactive landscape for creativity and innovation. In Proc. of the ACM Conf. On Human Factors in Computing Systems (CHI99), Pittsburgh, May 15-20, 1999, pp. 120-127.

[Szekely 96] Szekely P.: Retrospective and Challenges for Model-Based Interface Development, Computer-Aided Design of User Interfaces. In: Proceedings of CADUI'96, J. Vanderdonckt (eds), Presses Universitaires de Namur, 1996.

[Szekely 93] Szekely, P. et al.: Beyond Interface Builders: Model-based interfaces tools. In Proc. Of the ACM Conf. On Human Factors in Computing (INTERCHI 93), ACM Press, 1993, pp. 383-390.

[Tandler 01] Tandler, P.: Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices. In Pro. Ubicomp 2001, Springer Verlag, 2001, pp. 96-115.

[Thevenin 01] Thevenin, D.: Adaptation en Interaction Homme-Machine : le cas de la Plasticité. PhD thesis, Grenoble, 2001, 234 pages. http://iihm.imag.fr/publs/2001/

[Thevenin 99] Thevenin, D., Coutaz, J.: Plasticity of User Interfaces: Framework and Research Agenda. In: Proc. Interact99, Edinburgh, A. Sasse & C. Johnson Eds, IFIP IOS Press Publ., 1999, pp. 110–117.

[Vanderdonckt 95] Vanderdonckt, J.: Knowledge-Based Systems for Automated User Interface Generation; The TRIDENT Experience. RP-95-010, Fac. Univ. de N-D de la Paix, Inst. d'Informatique, Namur, B, 1995.

[Vanderdonckt 93] Vanderdonckt, J., Bodart, F., Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In Proc. INTERCHI 93, ACM Press, 1993, pp. 424-429.

[Want 01] Want, R., Shilit, B.: Expanding the Horizon of Location-Aware Computing, IEEE Computer, 34(8), 2001, pp. 31-34.

[Winograd 01] Winograd, T.: Architecture for Context, Human Computer Interaction, Lawrence Erlbaum Publ., 16(2-4), 2001, pp. 401-419.

[WP3.1 02] Application of tools to case studies, Cameleon Project WP3.1 working paper, V1, 2002.

[Zizi 94] Zizi, M., Beaudouin-Lafon, M. Accessing Hyperdocuments through Interactive Dynamic Maps, ECHT94, ACM Press, 1994, pp. 126-135.