Chapter 56

HCI and Software Engineering: the Case for User Interface Plasticity

Joëlle Coutaz, Gaëlle Calvary
Université Joseph Fourier, Grenoble
Laboratoire d'Informatique de Grenoble (LIG), BP 53, 38041 Grenoble Cedex 9
Tel. +33 4 76 51 48 54
{Joelle.Coutaz, Gaelle.Calvary}@imag.fr

1. Introduction

Human Computer Interaction (HCI) and Software Engineering (SE) are like two old friends with different backgrounds: they share values but they use them differently. Both domains address the design and development of useful and usable systems, and both are concerned with "requirements analysis", "incremental and iterative design", as well as "quality assurance". However, they address these problems with different development processes, different notations, and different priorities. For HCI, the Human is the first class entity in all phases of the development process. For SE, the final objective is a running system developed at minimal cost, and delivered in time, while satisfying contractual specifications. The user is, at best, involved at the very beginning of the process, and hopefully at the very end of the project for summative evaluation. However, this is too little, and too late, to avoid or correct wrong design decisions. Even in the early stages of the development process, functional requirements and quality goals are rarely the result of a close collaboration between HCI and SE specialists.

There are many reasons for the lack of collaboration between HCI and SE scientists and practitioners: mutual ignorance resulting from educational background, and from there, economic consideration. HCI methods such as contextual design (see Chapter XX), scenario-based approaches (see chapter XX), and task analysis (see chapter XX), are perceived as too demanding in terms of time and competence to inform the system functional specifications and non functional requirements in a formal and timely manner. On the other hand, UML use cases, which express the functions that the system should support with a scenario-based flavor, are pale attempts to factor out user-centered concerns. They do not result from a human centered requirements analysis nor do they have the expressive power of task models. Task model notations such as CTT [Paternò 03] or UAN [Hartson 90], which use LOTOS operators and logic, familiar to computer scientists, to express temporal dependencies and task composition, are not used by software engineers. Conversely, domain-dependent concepts referenced in task models are ill-defined, whereas UML class diagrams would improve task specifications significantly.

In summary, HCI and SE pursue the same goal, using development processes and notations that sometimes overlap, and that sometimes complement each other. In this chapter, we present one way to exploit both fields for the development of plastic user interfaces using the notion of *model* as the keystone between the two disciplines. In the following section, we present the problem space of User Interface (UI) plasticity followed, in Section 3, by three exemplars of plastic interactive systems that illustrate aspects of the problem space. We then introduce the key objectives and principles of Model Driven Engineering [Planet] (Section 4) and show, in Section 5, how they can be exploited as a solution space for UI plasticity.

2. The Problem Space of User Interface Plasticity

The term *plasticity* is inspired from the capacity of solids and biological entities such as plants and brain, to adapt to external constraints to preserve continuous usage. Applied to interactive systems, *UI plasticity is the capacity of user interfaces to adapt to the context of use while preserving usability* [Thevenin 99]. We define these terms in details in the following sections.

2.1 Context and Context of use

It is commonly agreed that *context* is about evolving, structured, and shared information spaces, and that such spaces are designed to serve a particular purpose [Coutaz 05]. In other words, context is not simply a state but part of a process (the purpose). Thus, there is no such thing as *the* context, but there is *a* context qualified by the

process it serves. This is why we use the term "context of use", and not simply "context", to refer to the information spaces that serve the adaptation process when context changes. A context change could be defined as the modification of the value of any element of the contextual information spaces. This definition would lead to an explosion of contexts. We need more structure. The following ontological foundation provides this structure.

2.1.1 Ontological Foundation for Context

As shown in Figure 1, a contextual information space is modeled as a directed graph where a node denotes a context and an edge a condition to move between two contexts. In turn, a context is a directed graph of situations where a node denotes a situation and an edge, a condition to move between two situations. Thus, a contextual information space is a two-level data structure, i.e., a graph of contexts where each context is in turn a graph of situations[1]. We need now to specify the domain of definition of contexts and situations.

A context is defined over a set E of *entities*, a set Ro of *roles* (i.e., functions) that these entities may satisfy, a set Rel of *relations* between the entities. Entities, roles and relations are modeled as expressions of *observables* that are captured and inferred by the system. For example, in a conference room, E denote the participants, Ro the roles of speaker and listener, and Rel, some spatial relations such as "in front of". The situations that pertain to the same context share the sets E, Ro and Rel.

The condition to move between two contexts is one of the following: E is replaced by a different set (e.g., the set E of participants is now replaced with the set of family members), Ro has changed (e.g., the roles of speaker and listener are replaced with that of parent), or Rel has changed (e.g., in addition to spatial relationships, temporal relationships between entities, now, matter).
The condition to move between two situations is one of the following:

- The cardinality of the set E has changed. For example, ten persons enter the room and are recognized by the system as participants (their observables match the characteristics and behavior of participants). If recognized as terrorists, then the system would detect a context change (and not a situation change).
- A role assignment to an entity has changed (e.g., participant *e* switches from the speaker role to the listener role),
- A relation between two entities has changed (e.g., participant *e* was in front of *e'*. Now, *e'* is in front of *e*).

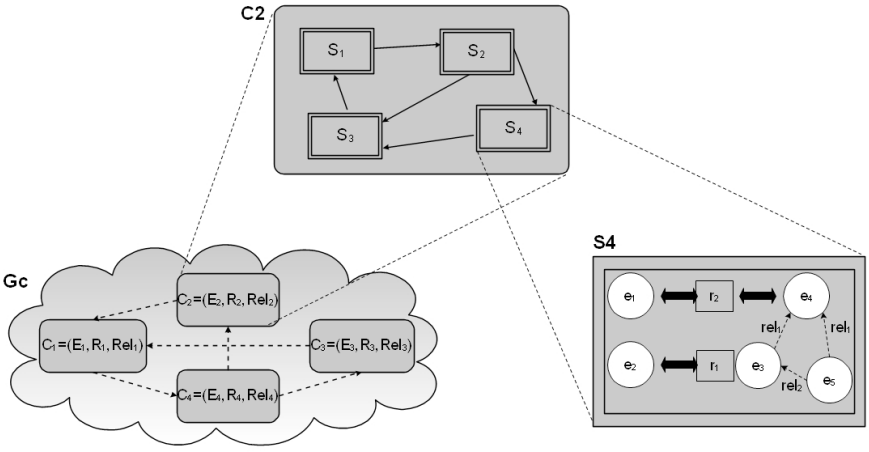


Figure 1. The graph of contexts Gc is composed of 4 contexts C1, C2, C3, C4 defined on their own sets of Entities, Roles, and Relations. In turn, Context C2 is composed of 4 situations S1, S2, S3, S4. By definition, these situations share the same sets of Entities, Roles, and Relations. In S4, Entities e1 and e4 (elements of E4) play the role r2 (element of R4), whereas Role r1 is played by Entities e2 and e3; e3 and e4 satisfy Relation rel1, e5 and e3 satisfy rel2, and e5 and e4 are related by rel1.

The ontology does not specify the nature of the entities, roles, relations, and observables. These are abstract classes from which a domain-dependent model can be specified. Using expression of observables, designers identify the set of Entities, Roles and Relations that are relevant for the case at hand. As discussed next, the observables of a context of use are organized into three information spaces.

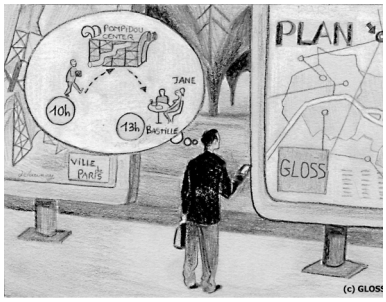
[1] If more structure is needed, situations may in turn be refined into "sub-situations", and so on.

2.1.2 Observables of the Context of Use

The observables of a context of use define three information spaces called the *user*, the *environment*, and the *platform* models.

- The *user model* denotes the attributes and functions that describe the archetypal person who is intended to use, or is actually using, the interactive system. This includes profile, idiosyncrasies, tasks and activities.
- The *environment* model includes attributes and functions that characterize the physical places where the interaction will take place, or is actually taking place. This includes numeric and/or symbolic locations (e.g., at home, in a public space, on the move in the street, in the train or car), social rules and activities, light and sound conditions.
- The *platform* model describes the computing, sensing, networking and interaction resources that bind together the physical environment with the digital world.

In the conventional GUI paradigm, the platform is limited to a single computing device, typically a workstation or a PDA, connected to a network and equipped with a fixed set of interaction resources such as screen, keyboard and stylus. Technological advances are leading to the capacity for individuals to assemble and mould their own interactive spaces from public hot spots and private devices to access services within the global computing fabric. Interactive spaces will also take the form of autonomous computing islands, or ecosystems, whose horizon will evolve, split and merge under the control of users. Figure 2 illustrates this view of ubiquitous computing where resources are coupled opportunistically to amplify human activities with new services, and where any real world object (e.g., a wall, pen and fingers) can play the role of an interaction resource. As a result, the platform should be viewed as a dynamic cluster of heterogeneous resources rather than a conventional mono-computing device.

(a) 

(b) 

Figure 2. Assembling interaction resources opportunistically: (a) Access to services within the global computing fabric via dynamic connection of a private device to a public hot spot (e.g., an active map). (b) Ecosystem: connecting two tablets to enlarge the screen real estate, or to start an intimate collaboration between users [Hinckley 03].

2.2 Usability

The term "usability" is interpreted in different ways by authors, even within the same scientific community. Usability has been identified with ease of use and learning, while excluding utility [Shackel 84, Nielsen 93]. In other cases, usability is used to denote ease of use and utility, while ignoring learnability. In software engineering, usability is considered to be an intrinsic property of the software product, whereas in HCI, usability is contextual: a system is not intrinsically usable or unusable. Instead, usability arises relatively to contexts of use.

The contextual nature of usability has been recently recognized by the ISO/IEC[2] 9126 standards developed in the software community with the overarching notion of "quality in use". Quality in use is "the capability of the software product to enable specified users to achieve specified goals with effectiveness, productivity, safety and satisfaction in specified contexts of use". Unfortunately, as shown in Figure 3, usability is viewed as one independent contribution to quality in use. Thus, the temptation is high for software people to assimilate usability to cosmetic issues limited to the user interface component of a software product, forgetting that system latency, reliability, missing functions and inappropriate sequencing of functions, have a strong impact on systems useworthiness.

Useworthiness is central to Cockton's argument for the development of systems that have value in the real world [Cockton 04, Cockton 05]. In value-centered approaches, software design should start from the explicit expression of an intentional creation of value for a selected set of target contexts of use. Intended value for target contexts are then

[2] ISO stands for International Organisation for Standardisation , IEC: International Electrotechnical Commission. The ISO/IEC 9126 series are part of the standards defined by the software engineering community.

translated into evaluation criteria. Evaluation criteria are not necessarily elicited from generic intrinsic features such as time for task completion, but are contextualized. They are monitored and measured in real usage to assess the achieved value.

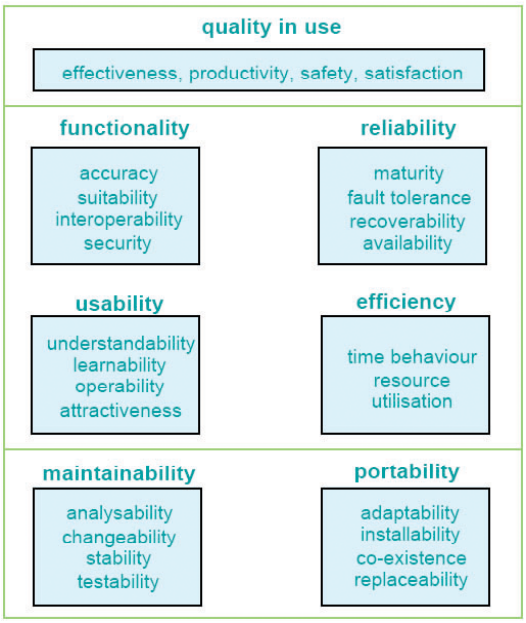


Figure 3. Usability model from ISO/IEC 9126-1.

Building on Cockton's approach, we suppose that for each of the target contexts of use C_i of a system, an intended value V_i has been defined, and that V_i has been translated into the set of triples $\{(c_{i1}, d_{i1}, w_{i1}), \ldots, (c_{ij}, d_{ij}, w_{ij}), \ldots (c_{in}, d_{in}, w_{in})\}$ where c_{ij} is an evaluation criteria, and d_{ij} and w_{ij}, the expected domain of values and relative importance of c_{ij} in C_i. As discussed above, c_{ij} may be a generic measurable feature or a customized measure that depends on the intended value in C_i. Usability U_i of the system for context C_i is evaluated against a combining function F_i on the set $\{(c_{i1}, d_{i1}, w_{i1}), \ldots, (c_{ij}, d_{ij}, w_{ij}), \ldots (c_{in}, d_{in}, w_{in})\}$ whose results is intended to lie within a domain of values D_i.

Coming back to the notion of plasticity, an interactive system S is *plastic from a source context of use C_i to a target of use C_j* if the following two conditions are satisfied:

- Adaptation, if needed, is supported when switching from C_i to C_j,
- Usability (value) is preserved in C_j by the adaptation process.

In turn, usability in the target context C_j is preserved if the following two conditions are satisfied:

- the usability function F_j defined for C_j lies within its intended domain D_j.
- the usability function *meta-*F_{ij} of the *meta-UI* lies within its intended domain *meta-*D_{ji} when transiting from C_i to C_j.

A meta-UI is to ambient computing what the desktop metaphor is to conventional workstations. It binds together the activities that can be performed within an interactive space. In particular, it provides users with the means to configure, control and evaluate the adaptation process. It may, or may not, negotiate the alternatives for adaptation with the user. A meta-UI without negotiation makes observable the state of the adaptation process, but does not allow the user to intervene. The system is autonomous. A meta-UI incorporates negotiation when, for example, it cannot make sound decisions between multiple forms of adaptation, or when the user must fully control the outcome of the process. The balance between system autonomy and too many negotiation steps depends on the case at hand.

The domain of plasticity of a system is the set C of contexts of use C_i for which usability is achieved.

We have defined usability by reasoning at the context level. If needed, a finer grain of reasoning can be applied at the situation level: intended value is defined for each situation of each context, then translated into evaluation criteria. Preserving usability is then evaluated on situation changes.

These definitions provide a theoretical framework where value comes first and defined on a per-context (or

situation) of use basis. For each of the intended target contexts (or situations), value is operationalized into a mix of generic and customized metrics. The problem is the identification of the relevant contexts of use and situations as well as the appropriate translation of value into significant metrics. We have no answer for operationalizing value, except to use generic measures when applicable, to instrument the system appropriately using sound software development techniques (such as AOP [Elrad 01]), and to apply a healthy dose of common sense. On the other hand, our ontological framework on context and its associated method presented in [Rey 05][3], can be used to define the boundaries of contexts and situations of use as well as their relationships. For our notion of context of use, the fundamental entities are the user(s), environment, and platform, each of them being characterized by observables monitored by the system. Section 5.1 shows how to integrate the monitoring of observables within the software architecture of an interactive system.

2.3 System Adaptation

System adaptation to context of use can take multiple forms. Here, we limit the discussion to the consequence of adaptation as perceived by users at the user interface. Adaptation can use one, or a combination of, the following techniques: UI remoulding, UI distribution, and UI migration.

2.3.1 UI remoulding

UI remoulding denotes the reconfiguration of the user interface that results from the application of one or several transformations on all, or parts, of the user interface. These transformations can be applied at multiple levels of abstraction, and they can be intra-modal, inter-modal, or multi-modal. These transformations include:

- *Suppression* of the UI components that become irrelevant in the new situation/context;
- *Insertion* of new UI components to provide access to new services relevant in the new situation/context;
- *Substitution* of UI components when UI components are replaced with new ones. Substitution can be viewed as a combination of suppression and insertion of UI components;
- *Reorganization* of UI components by revisiting their spatial layout and/or their temporal dependency. Reorganization may result from the suppression, insertion, or substitution of UI components. On the other hand, switching from a portrait to a landscape view requires spatial reorganization only.

Remoulding is *intra-modal* when the source UI components concerned by the transformations are retargeted within the same modality. Remoulding is *inter-modal* if the source UI components expressed in one modality (say GUI) are transformed into UI components using a different modality (say speech). Remoulding is *multi-modal* if it uses a combination of intra- and inter-modal transformations. For example, Teresa supports multi-modal remoulding [Berti 05].

The transformations can be performed at multiple levels of abstraction:

- At the *Physical Presentation (PP) level*, physical interactors (widgets) used for representing functions and concepts are kept unchanged but their rendering and behaviour may change. For example, if a concept is rendered as a button class, this concept is still represented as a button in the target UI. However the look and feel of the button or its location in the workspace may vary. This type of adaptation is used in Tk as well as in Java/AWT with the notion of peers.
- At the *Logical Presentation (LP) level*, adaptation consists of changing the representation of functions and concepts. For example, the concept of month can be rendered as a Label+Textfield, or as a Label+Combobox, or as a dedicated physical interactor. In an LP adaptation, physical interactors can replace each other provided that their representational and interactional capabilities are equivalent. The implementation of an LP level adaptation can usefully rely on the distinction between Abstract Interactive Objects and Concrete Interactive Objects as presented in [Vanderdonckt 93]. Changes at the LP level imply changes at the PP level.
- At the *Dialog Component (DC) level*, the tasks that can be executed with the system are kept unchanged but their organisation is modified. As a result, the structure of the dialogue structure is changed. AVANTI's polymorphic tasks [Stephanidis 01] are an example of a DC level adaptation. Changes at the DC level imply changes at the LP and PP levels.
- At the *Functional Core Adaptor (FCA) level*, the nature of the entities as well as the functions exported by the functional core (the services) are changed. Zizi's semantic zoom is an example of an FCA level adaptation [Zizi 94]. Changes at the FCA level imply changes at the DC, LP, and PP levels.

2.3.2 UI distribution

[3] Editorial Note: this thesis is written in French. If needed, a summary of the method could be presented in an annex of the chapter.

A UI is distributed when it uses interaction resources that are distributed across a cluster. For example, in graphical UI's (GUI), the rendering is distributed if it uses surfaces that are managed by different computing devices. Distribution is *static* when it is performed off-line between sessions. It is *dynamic* when it occurs on the fly.

The granularity of UI distribution may vary from application level to pixel level:

- At the *application level*, the GUI is fully replicated on the surfaces managed by each computing device. The x2vnc implementation of the VNC protocol offers an application level distribution.
- At the *workspace level*, the unit for distribution is the workspace. A workspace is a logical space that supports the execution of a set of logically connected tasks[4]. PebblesDraw [Myers 01] and Rekimoto's Pick and Drop [Rekimoto 97] are examples of UI distribution at the workspace level.
- The *interactor level* distribution is a special case of the workspace level where the unit for distribution is an elementary interactor.
- At the *pixel level*, any user interface component can be partitioned across multiple surfaces. For example, in the DynaWall [Streitz 99], a window may simultaneously lie over two contiguous white boards as if these were managed by a single computer.

2.3.3 UI Migration

UI migration corresponds to the transfer of all or part of the UI components to different interaction resources whether these resources belong to the current platform or to another one. Migration is *static* when it is performed off-line between sessions. It is *dynamic* when it occurs on the fly. In addition, the migration of a user interface is *total* if the user interface moves entirely to a different platform (application level migration). It is *partial* (at the workspace, interactor or pixel levels) when a subset only of the user interface moves to different interaction resources. For example, on the arrival of a PDA, the control panels currently rendered on a whiteboard migrate to the PDA (workspace level migration).

Migration and distribution are two distinct notions: a UI may be distributed but not migratable (static distribution). A centralized UI may migrate, but if the migration is total, it remains centralized on the target platform.

2.3.4 State Recovery

The granularity of state recovery characterizes the effort users must apply to carry on their activity after adaptation has occurred. State recovery can be performed at the session, task, and physical action levels:

- When the system state is saved at the session level, users have to restart the interactive system from scratch. They rely on the state saved by the service (functional core) level before adaptation is taking place.
- At the task level, the user can pursue the job from the beginning of the current interrupted task (provided that the task is attainable in the retargeted system).
- At the physical action level, the user is able to carry on the current task at the exact point within the current task (provided that the task is attainable in the new version of the user interface).

The examples presented next illustrate the problem space of plastic UI's.

3. Case Studies

The home heating control system, CamNote, and Sedan-Bouillon are three examples of plastic interactive systems developed according to the MDE approach presented Sections 4 and 5. The services they provide are accessible from different types of computing devices including workstations, personal digital assistants (PDA), and mobile phones. The UI of the heating control system is kept centralized on one single device at a time, whereas for CamNote and the Sedan-Bouillon web site, the UI components can be dynamically distributed and migrated across the interaction resources currently available in the interactive space. CamNote and Sedan-Bouillon differ in the technological spaces used for implementation: CamNote is Java-centric whereas Sedan-Bouillon uses PHP-MySQL Internet solutions.

---

[4] A workspace is analogous to the concept of *focus area* used in Contextual Design for expressing the user environment design.

3.1 Home heating Control System

The home heating control system envisioned is intended to be used (a) at home through a dedicated wall-mounted device or through a PDA connected to a wireless home-net; (b) in the office, through the Web, using a standard work station; (c) anywhere using a mobile phone or a watch.

A typical user's task consists of consulting and modifying the temperature of a particular room. Figures 4 and 5 show versions of the system for a home comprised of two rooms:

- In Figure 4a, the screen size is comfortable enough to display the state of all of the home thermostats.
- In Figure 4b, 4c, and 4d, the screen size allows the rendering of the state of one single thermostat at a time. Thus, an additional navigation task is required to access the state of the second thermostat. In 4b, the task is supported by a button, whereas in 4c and 4d, a combo-box and hyperlinks are used to browse the system state. In 4e, the screen of the watch is so small that the user interface is limited to the living room, the most important place of the home.

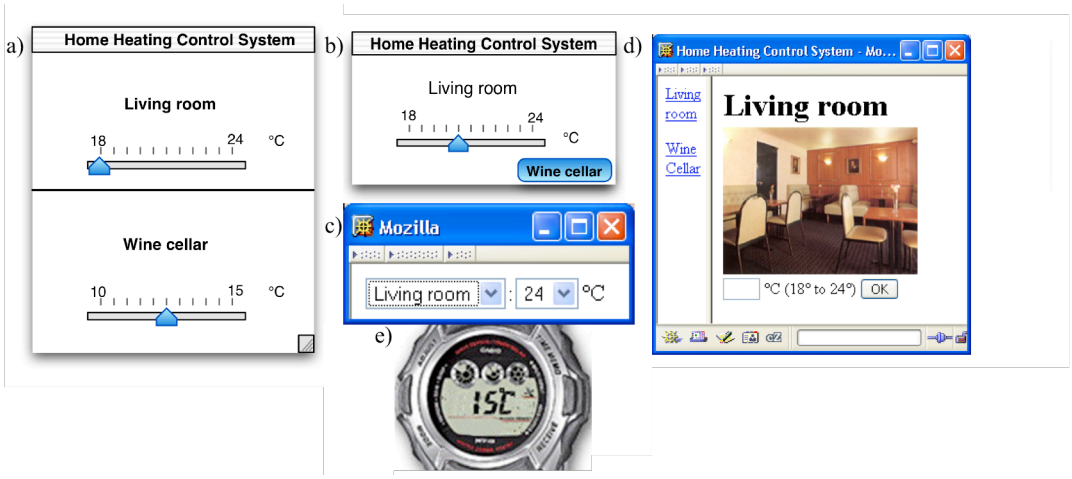


Figure 4. Home heating Control system. a) Large screen. The temperature of the rooms are available at a glance. b) c) d) Small screen: the temperature of a single room is displayed at a time but the system state is modifiable and browsable via a button (in b), combo-box (in c) and hyperlinks (in d). e) Very small screen: only one single room is observable. Temperature setting and navigation tasks are not available.

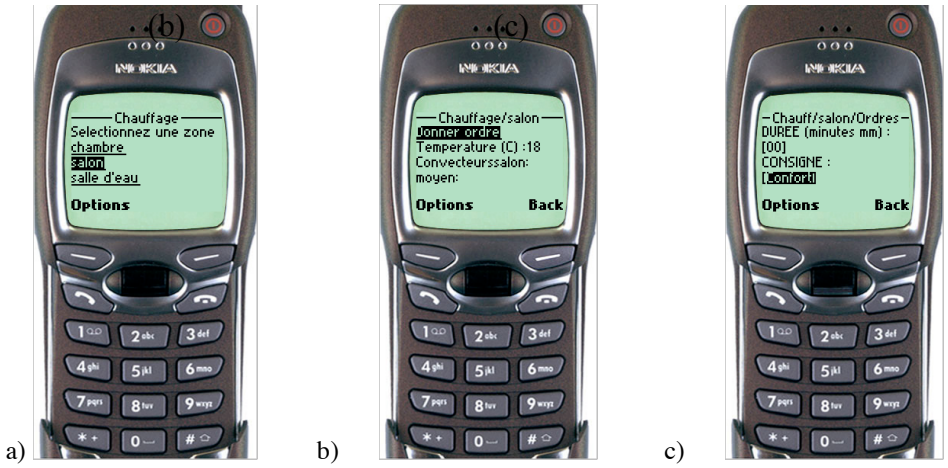


Figure 5. Modifying the temperature using a mobile phone.

Figure 5 shows the interaction trajectory for setting the temperature of a room with a mobile phone.

- In 5a), the user selects the room (e.g., "le salon" – the living room).
- In 5b), the system shows the current temperature of the living room.
- By selecting the editing function ("donner ordre"), one can modify the temperature of the selected room (5c).

When comparing with the situation depicted in Figure 4, two navigation tasks (i.e., selecting the room, then selecting the edit function) must be performed in order to reach the desired state. In addition, a title has been added to every page to recall the user with the current location within the information space.

Using the taxonomy of Section 2, the home heating control system can be characterized in the following way: the context of use is limited to the user's preference (what is themost important room in the home), and to the platform. This platform is always a mono-computing device. Context of use can be modeled as a graph of situations where the set of entities is the user and the intended computing devices. The sets of Roles and Relations are empty. Switching between two situations occurs when using a different computing device. Adaptation to context of use boils down to adapting the UI to the resources of the currently used computing device. Adaptation relies on remoulding only. It is GUI intra-modal and the grain of recovery is the session: switching between devices on the fly requires starting from scratch. The adaptation level depends on the source and target platforms: when switching between the b-c-d- situations, adaptation occurs at the Logical Presentation level: a widget is replaced by a functionally equivalent widget. When switching between the large screen and the small screen, the task space is not changed but organized differently: adaptation occurs at the Dialogue Controller level. When switching to the watch, a number of services are no longer available: the adaptation is performed at the FCA level.

3.2 CamNote

CamNote (for CAMELEON Note) is a slides viewer that runs on a dynamic heterogeneous platform. This platform may range from a single PC to a cluster composed of a PC and a PDA. Its UI is structured into four UI workspaces: a slides viewer, a note editor for associating comments to slides, a video viewer also known as "mirror pixels" that shows a live video of the speaker, and a control panel to browse the slides and to setup the level of transparency of the mirror. The mirror is combined with the slides viewer using alpha-blending. Speakers can point at items on the slide using their finger. This means of pointing is far more compelling and engaging than the conventional mouse pointer that no one can see.

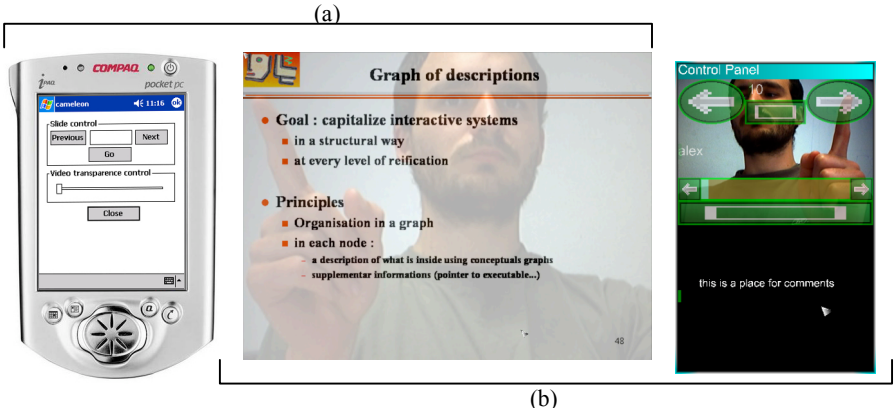


Figure 6. The user interface of CamNote. (a) The UI of CamNote when distributed on a PC and a PocketPC screens; (b) the control panel when displayed on the PC screen.

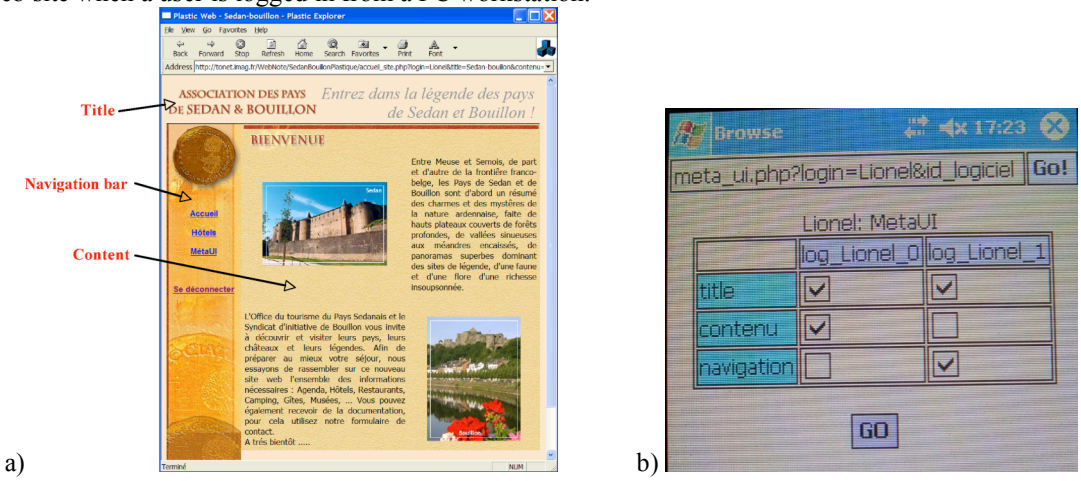
Picture 6a shows a configuration where the graphical UI is distributed across the screens of a PC and of a PDA. The slides viewer is displayed in a rotative canvas so that it can be oriented appropriately when projected on a horizontal surface. If the PDA disappears, the control panel automatically migrates to the PC screen. Because different resources are now available, the control panel includes different widgets, but also a miniature representation of the speaker's video is now available. During the adaptation process, users can see the control panel emerging progressively from the slides viewer so that they can evaluate the progress of the adaptation. The UI, which was distributed on a PC and a PDA is now centralized on the PC (Picture 6b). Conversely, if the PDA re-enters the interactive space, the UI automatically switches to the configuration of Figure 6a, and the control panel disappears from the PC screen by weaving itself into the slides viewer before reappearing on the PDA.

In this exemplar, context of use is limited to the platform and the transitions between the situations occur on the arrival/departure of a computing device. Adaptation is based on migration and distribution at the workspace level, as well as on intra-modal GUI remoulding at the Dialogue Controller level: when the control panel resides on the PDA, the note editing task is no longer available. Adaptation is automatic: the user has no control over the adaptation process, but a minimum of meta-UI exists to express the transition between two situations (the

weaving effect). State recovery is performed at the physical action level: the slides show is not disturbed by adaptation.

3.3 The Sedan-Bouillon Web Site

"Sedan-Bouillon" is a web site that aims at promoting tourism in the regions of Sedan (France) and Bouillon (Belgium) (http://www.bouillon-sedan.com/). It provides tourists with information for visiting and sojourning in these regions including a selection of hotels, camping, and restaurants. Figure 7a shows a simplified version of this web site when a user is logged in from a PC workstation.



a) b)

Figure 7 – The Sedan-Bouillon web site. (a) UI centralized on a PC screen. (b) The control panel of the meta-UI to distribute UI workspaces across the resources of the interactive space. The lines of the matrix correspond to the workspaces, and the columns denote the browsers currently used by the same user.

Preparing a trip for vacation is an exciting experience when shared by a group of people. However, one single PC screen does not necessarily favour collaborative exploration. By dynamically logging to the same web site with a PDA, users are informed on the PDA that they can distribute the UI components of the site across the interaction resources currently available. In the example of Figure 7b, the user asks for the following configuration: the title must appear on the PDA as well as on the PC (the title slots are ticked for the two browsers available), whereas the content should stay on the PC and the navigation bar should migrate to the PDA. Figure 8 shows the resulting UI. At any time, the user can ask for a reconfiguration of the UI by selecting the "meta-UI" link in the navigation bar. The UI will be reconfigured accordingly.

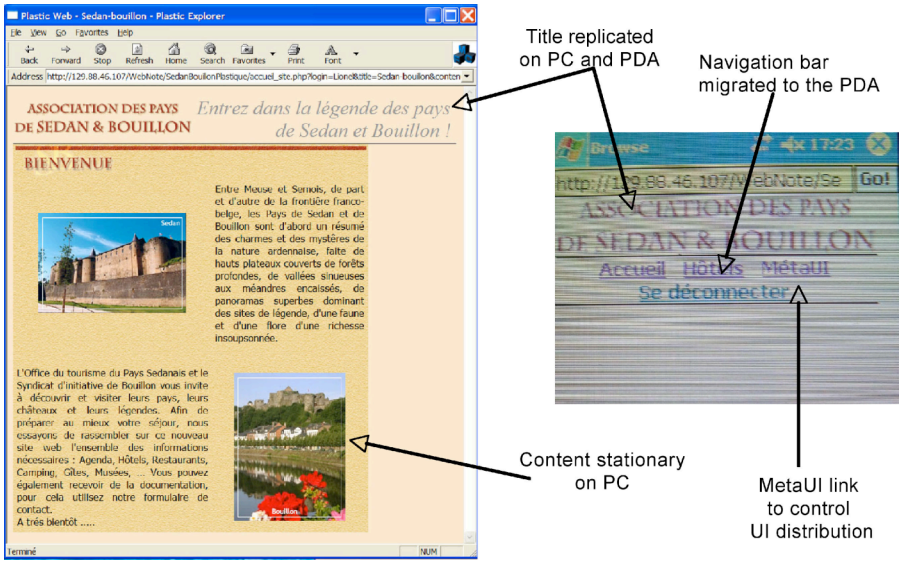


Figure 8- The Sedan-Bouillon web site when distributed across the resources of the interactive space. The MetaUI link allows users to return to the configuration panel shown in Figure 7b.

In terms of the plasticity problem space, the Sedan-Bouillon web site is very similar to CamNote: same model of context of use, adaptation based on distribution and migration at the workspace level, and GUI intra-modal remoulding at the workspace level. Contrary to CamNote, remoulding is performed at the Logical Presentation level (no task is suppressed nor restructured), and state recovery is supported at the task level: if adaptation occurs as the user is filling a form, the form content is lost by the adaptation process. Contrary to CamNote, the user has full control over the reconfiguration of the UI using the control panel of the meta-UI.

Having characterized three exemplars in the problem space, we now consider the method and mechanisms necessary to support interactive systems plasticity. We advocate a Model Driven Engineering (MDE) approach.

4. Model Driven Engineering

The motivation for MDE is the integration of knowledge and techniques developed in software engineering using the notions of model and model transformation and mapping as the key concepts.

In the early days of computer science, software systems were simple programs written in assembly languages. In those days, a code-centric approach to software development was good enough, not to say unavoidable to ensure a fine control over the use of computing resources. Over the years, the field has evolved into the development of distinct paradigms and application domains leading to the emergence of multiple Technological Spaces (TS).

"*A technological space is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities.*" [Kurtev 2002]. Examples of technological spaces include documentware concerned with digital documents using XML as the fundamental language to express specific solutions, dataware related to data base systems, ontologyware, … HCIware!

Today, technological spaces can no longer evolve in autarky. Most of them share challenges of increasing complexity, such as adaptation, to which they can only offer partial solutions. Thus, we are in a situation where concepts, approaches, skills, and solutions, need to be combined to address common problems. This is where MDE comes into play. MDE aims at achieving integration by defining gateways between technological spaces using a model-based approach. The hypothesis is that models, meta-models, models transformations and mappings, are everything.

4.1 Models

A model is a representation of a thing (e.g., a system), with a specific purpose. It is "able to answer specific questions in place of the actual" thing under study [Bezivin 01]. Thus, a model, built to address one specific aspect of a problem, is by definition a simplification of the actual thing under study. For example, a task model is a simplified representation of some human activities (the actual thing under study), but it provides answers about how "representative users" proceed to reach specific goals.

A model may be *physical* (a tangible entity in the real world), *abstract* (in human mind), or *digital* (within computers) [Favre 04a, Favre 04b]. For example, considering the person Peter, a printed photograph of Peter is a physical representation of Peter that his mother (for example) uses for specific purpose. Peter's mother has mental representations of him as a good son, or as a brilliant researcher (multiple abstract models about Peter). The authentification system that runs on Peter's computer knows him as a login name and password (digital model). If Peter's portrait is digitized as a JPEG picture, then the JPEG file is a digital model of a physical model. When displayed on the screen, the JPEG file is transformed into yet another digital graphics model in the system's main memory before being projected on the screen as an image (yet another physical model that Peter's mother can observe).

As this example shows, models form oriented graphs (μ graphs) whose edges denote the μ relation "is represented by". In other words, a model can represent another model, and a model can be represented by several models (see Figure 9).

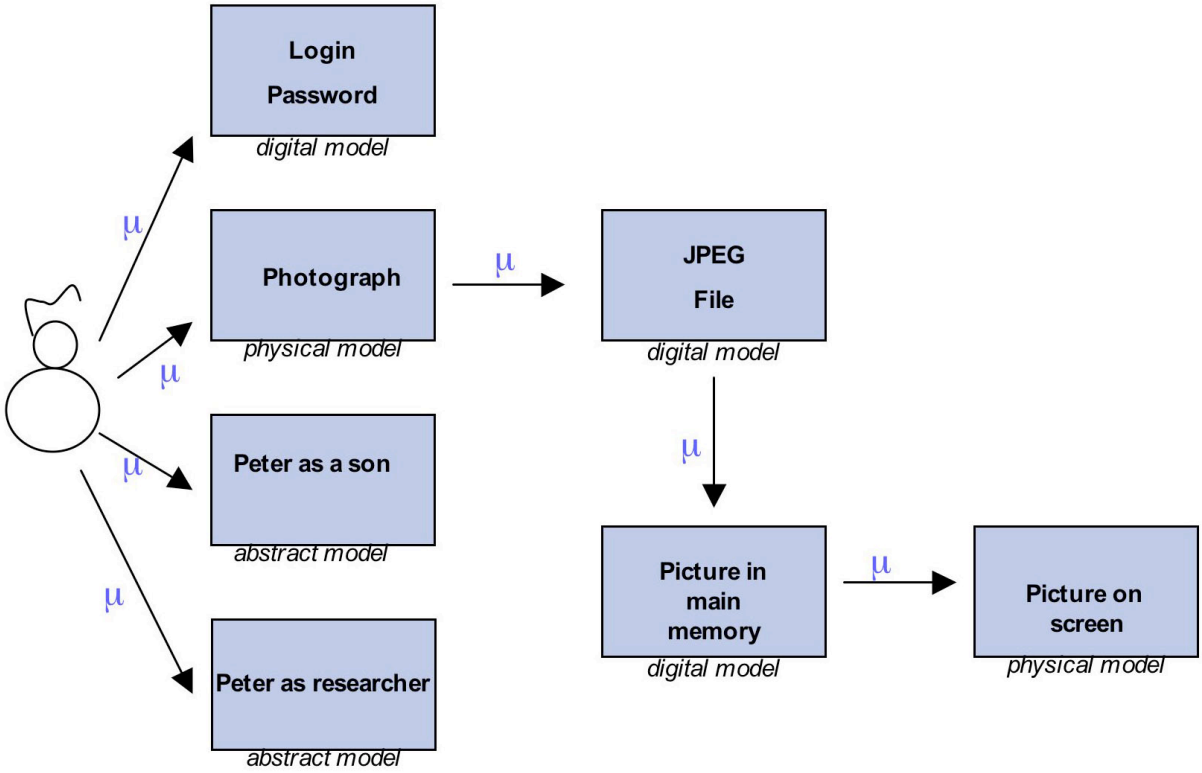


Figure 9. Models organized as oriented μ graphs.

Fypically, scenarios developed in HCI [Rosson 02] are contemplative models of human experience in a specified setting.

In order to be processed (by humans, and/or by computers), a model must comply with some shared syntactic and semantic conventions: it must be a well-formed expression of a language. This is true both for productive and contemplative models: most contemplative models developed in HCI use a mix of drawings and natural language. A language is the set of all well-formed expressions that comply with a grammar (along with semantics). In turn, a grammar is a model from which one can produce well-formed expressions (or models). Because a grammar is a model of a set of models, it is called a meta-model.

4.2 Meta-model

A meta-model is a model of a set of models that comply with it. It sets the rules for producing models. It does not represent models. Models and meta-models form a χ tree: a model complies to a single meta-model, whereas a meta-model may have multiple compliant models.

As an illustration, suppose that the authentification system mentioned above is a Java program J. J is a digital model that represents Peter and that complies with the Java grammar G_J. G_J does not represent J, but defines the compliance of J with Java. G_J is one possible meta-model, but not the only one. The authentification system could also be implemented in C (yet another digital model of Peter). It would be compliant with the C grammar G_C. Grammars G_C and G_J could in turn, be produced from the same grammar such as EBNF. EBNF[5] is an example of a meta-metamodel.

A *meta-metamodel is a model of a set of meta-models that are compliant with it*. It does not represent meta-models, but sets the rules for producing distinct meta-models. As shown in Figure 10, the OMG Model-Driven Architecture[6] (MDA) initiative has introduced a four-layer modeling stack as a way to express the integration of a large diversity of standards using MOF (Meta Object Facility) as the unique meta-metamodel. EBNF, G_J and G_C, the Java and C programs are models that belong to the programming technological space. Within the MDA

[5] EBNF (Extended Backus-Naur Form) is defined as the s ISO/IEC 14977:1996 standard

[6] MDA is a specific MDE deployment effort around industrial standards including MOF, UML, CWM, QVT, etc.

technological Space, the java source code of our authentification system becomes a UML Java model compliant with the UML meta-model. In the XML technological space, the java source code could be represented as a JavaML document compliant with a JavaML DTD[7].

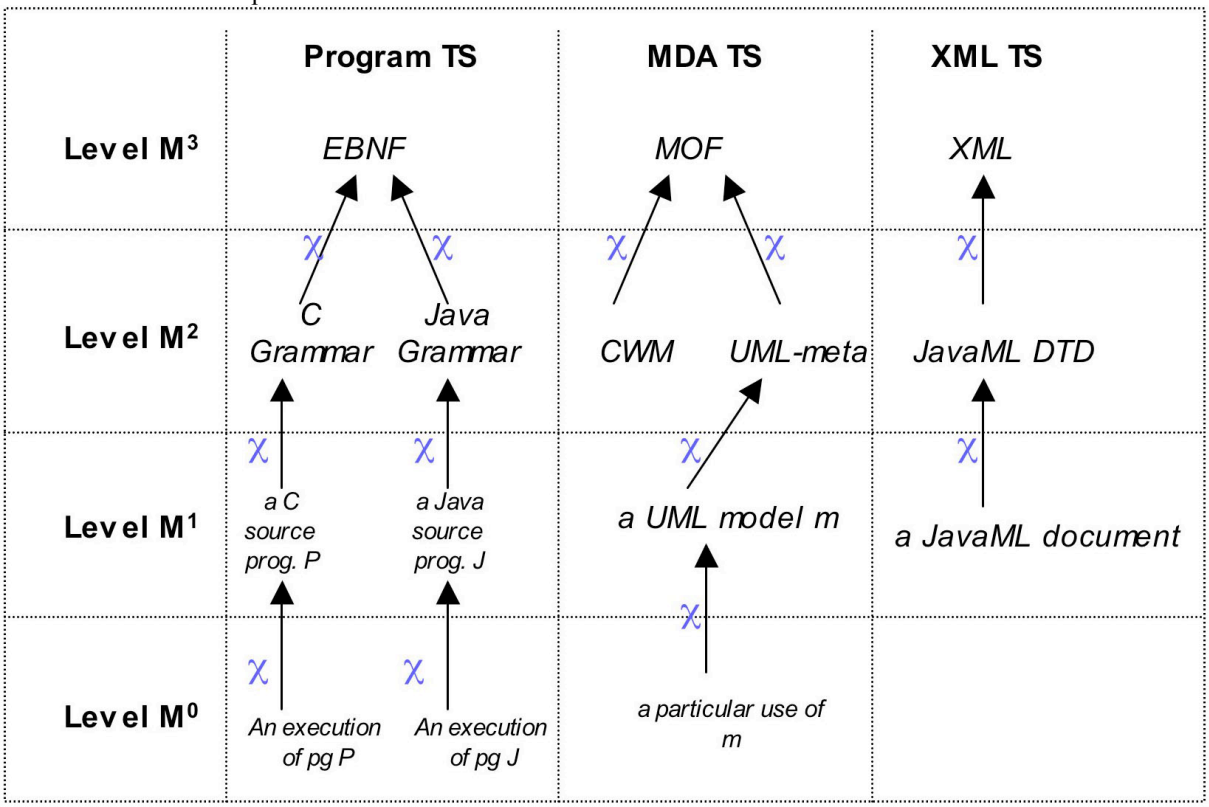


Figure 10. The OMG MDA four-layer modeling stack.

As shown in Figure 10, μ relations ("is represented by") and χ relations ("complies with") make explicit the multiplicity of existing technological spaces as well as their systematic structure into 3 levels of modeling spaces (the so-called M1, M2, M3 levels of MDA)+the M0 level that corresponds to a system, or parts of a system. The μ and χ relations, however, do not tell how models are produced within a technological space nor how they relate to each other across distinct technological spaces. The notions of *transformation* and *mapping* is the MDE answer to this issue.

4.3 Transformations and Mappings

In the context of MDE, *a transformation is the production of a set of target models from a set of source models, according to a transformation definition*. A transformation definition is a set of transformation rules that together describe how source models are transformed into target models [Mens 04]. Source and target models are related by the τ relation "is transformed into". Note that a set of transformation rules is a model (a transformation model) that complies with a transformation meta-model.

Relation τ expresses an overall dependency between source and target models. However, experience shows that finer grain of correspondence needs to be expressed. Typically, the incremental modification of one source element should be propagated easily into the corresponding target element(s) and vice versa. The need for traceability between source and target models is expressed as *mappings* between source and target elements of these models. For example, each task of a task model and the concepts involved to achieve the task, are rendered as a set of interactors in the Concrete User Interface model. Rendering is a transformation where tasks and their concepts are mapped into a workspace which in turn is mapped into a window populated with widgets. The correspondence between the source task (and concepts) and its target workspace, window and widgets, is maintained as a mapping function. In section 5, we will see how mapping functions can be exploited at run time.

[7] DTD (Document Type Definition). In the XML Technological spaces, a DTD defines the legal building blocks of an XML document.

Transformations can be characterized within a four-dimension space:

- The transformation may be *automated* (it can be performed by a computer autonomously), it may be *semi-automated* (requiring some human intervention), or it may be *manually performed* by a human. For example, given our current level of knowledge, the transformation of a "value-centered model" into a "usability model" can only be performed manually. On the other hand, User Interface generators such as CTTE [Mori 02, Mori 04] produce user interfaces automatically from a task model.

- A transformation is *vertical* when the source and target models reside at different levels of abstraction. UI generation is a vertical top down transformation from high-level descriptions (such as a task model) to code generation. Reverse engineering is also a vertical transformation but it proceeds bottom up, typically from executable code to some high-level representation by the way of abstraction. A transformation is *horizontal* when the source and target models reside at the same level of abstraction. For example, translating a Java source code into C, preserves the original level of abstraction.

- Transformations are *endogenous* when the source and target models are expressed in the same language. They are *exogenous* when sources and targets are expressed in different languages while belonging to the same technological space. For example the transformation of a Java source code program into a C program is exogenous.

- When crossing technological spaces (e.g., transforming a java source code into a JavaML document), then additional tools (exporters or importers) are needed to bridge the gap between the spaces. *Inter-technological transformations* are key to knowledge and technical integration. This is the quest of MDE.

In the following section, we show how the MDE principles can be applied to the development of plastic interactive systems by bringing together HCIware with mainstream software technological spaces.

5. MDE for Plastic User Interfaces

There are several approaches for the development of the systems presented in Section 3. For example, the user interface of the home heating control system can be produced based on a sound iterative user-centered design process. A similar process can be conducted in parallel for the mobile phone version. In the absence of explicit links between the two streams of development, this approach is doomed to failure: inconsistencies between the two user interfaces will occur, and maintenance of the two versions will be harder to synchronize. In addition, development efforts will be duplicated.

Alternatively, the UI code for the workstation can serve as a reference for transformations into a new target UI for the mobile phone. Duplication of efforts is avoided and the approach is technically attainable. For example, using XLST, the HTML UI produced for the workstation can easily be translated into a WML UI for the mobile phone. Crossing technological spaces at the code level is not as straightforward, but is feasible. However this approach does not address deep UI adaptations: a code-centric approach is certainly able to address UI adaptation at the PP and LP levels (Physical and Logical Presentation). It cannot cope with higher levels of adaptation such as DC and FCA levels (Dialogue Control and Functional Core levels) where, for example, task sequencing may need substantial reorganization.

UI migration and UI distribution introduce new challenges. In a code-centric approach, the code of CamNote and Sedan-Bouillon Web site would explicitly include instructions that make reference to the context of use. For example, "If am running on the PDA, then my rendering is *this* ; if I am running of a workstation, then my rendering is *that*". Clearly, this approach is acceptable for concept demonstration. It does not scale up in the context of ubiquitous computing where opportunism is paramount. This short analysis militates for an *approach that continuously exploits models at multiple levels of abstraction not only for the development process but for the run time phase as well*.

In order to define the appropriate models, we need first, to investigate the functions that plastic interactive systems must support. Having identified the set of functions, we can then attach a useful set of perspectives (models) to each function.

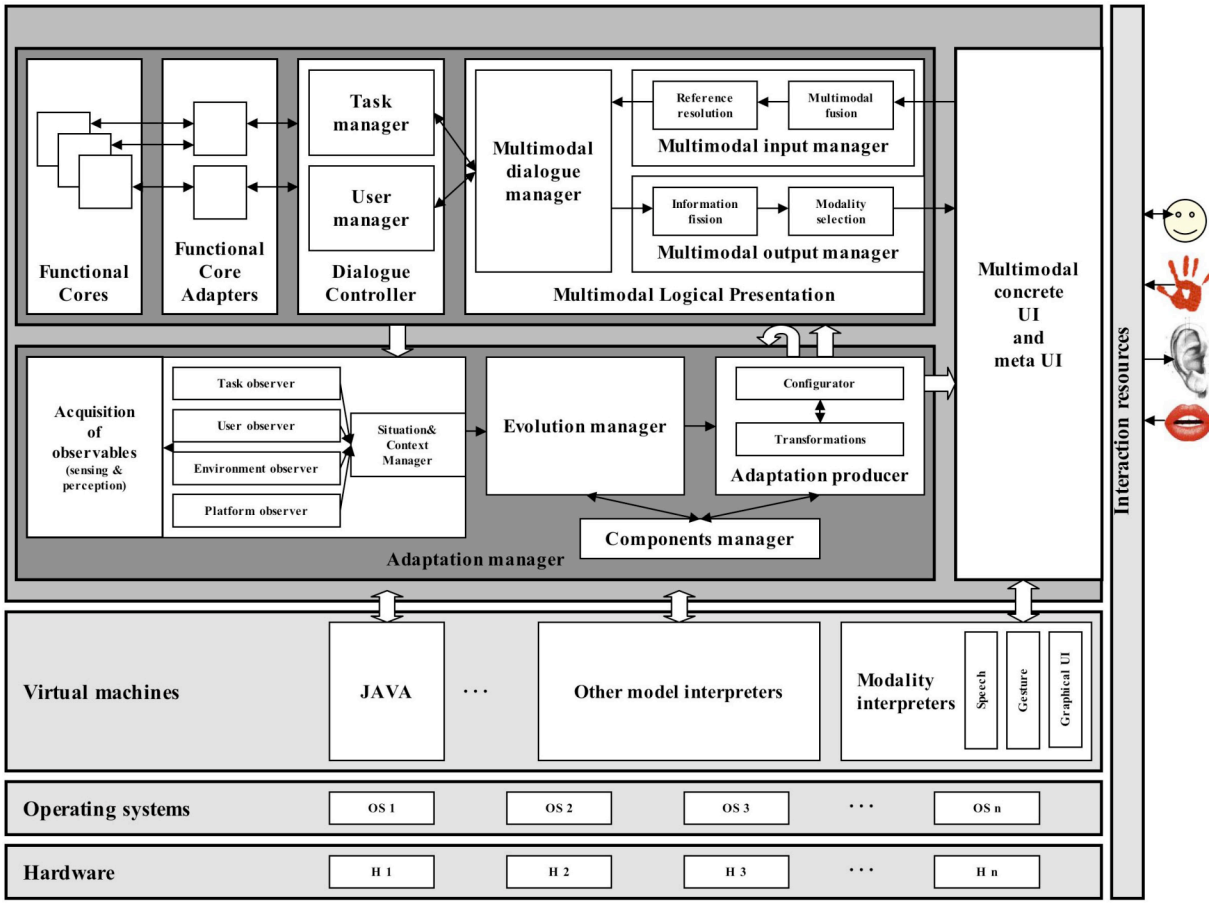


Figure 11. A reference model for the functional decomposition of plastic interactive systems.

5.1 Functional Decomposition for Plastic Interactive Systems

The picture shown in Figure 11 is a reference model that makes explicit the functional decomposition of an interactive space for supporting plasticity [Balme 04]. At the bottom of the picture, the hardware, a dynamic assembly of processors, sensors and actuators, as well as an heterogeneous set of *interaction resources* such as pointing devices, microphones, private-eyes, and large-size screens. Each computer executes its own *operating system* and a diversity of *virtual machines*.

The virtual machines are *model interpreters* of various sorts:

- Java VM is a model interpreter where the model is a byte-code program.
- *Modality interpreters* denote conventional GUI or Post-WIMP systems, speech processing as well as text-to-speech processors. They interpret models that are modality specific.
- *Other model interpreters* represent any set of interpreters used at run time including UI transformers from high level specifications.

The virtual machines, operating systems, and hardware (including the interaction resources) define the effective platform. The rest of the figure represents the interactive system per se. The interactive system is comprised of two functional complementary facets:

- the fundamental functions that motivate the existence of the system,
- the adaptation manager so that these functions can be adapted to the context of use.

The fundamental functions are structured according to the five abstraction layers of the Arch reference model [Arch 92]: the Functional Core and the Functional Core Adapter, the Dialogue Controller, the Logical and Physical Presentation layers denoted in the picture as the Multimodal Logical Presentation and Multimodal Concrete UI to stress the multimodal dimension of user interfaces (see Chapter XX in this book for more information on multimodal interactive systems)[8].

[8] Editorial note: additional description of Arch can be provided here if not presented elsewhere in the book.

By acting on input interaction resources, users (represented on the figure as eyes, hands, lips, and hears), generate low-level events that are processed by the appropriate modality interpreter(s), then fed into the Multimodal Logical Presentation layer. The *Multimodal Logical Presentation* is in charge of multimodal fusion and fission both for input and output at the appropriate level of abstraction. As in the Arch model, the *Dialogue Controller* controls the domain and user-dependent task sequencing, and the *Functional Core Adapters* express the software interfaces to accommodate mismatches between the domain-specific *Functional Cores* (applications) and the user interface per se of the interactive space. This five-level *Arch-based functional structure* has the capacity to self-adapt to the context of use thanks to the Adaptation Manager.

The *Adaptation Manager* aims at satisfying three functional requirements:

1- To acquire the context of use at the appropriate level of abstraction, and to detect (and signals) context and situation changes (due, for example to the arrival of a PDA): this is performed by the *Context&Situation Manager*.
2- To elaborate a reaction in response to a new situation/context: this is performed by the *Evolution Manager*. For example, "if a new PDA arrives, then move the control panels of the interactive system to the PDA."
3- To execute the reaction plan: this is performed by the *Adaptation Producer*.

The *Context&Situation Manager* is structured into four levels of abstraction [Coutaz 05]: at the lowest level, the system's view of the world is provided by a collection of sensors. Sensors may be physical sensors such as RFID's, or software sensors that probe a user's identity and the state of the platform cluster. The *sensing layer* generates numeric observables. To determine meaning from numeric observables, the Context&Situation Manager must perform transformations. The *perception layer* is independent of the sensing technology and provides symbolic observables at the appropriate level of abstraction. The *situation and context identification layer* identifies the current situation and context from observables, and detects conditions for moving between situations and contexts. Services in this layer specify the appropriate entities, roles and relations for operating within the user's activities. They can be used to predict changes in situation or in context, and thus anticipate needs of various forms (system-centric needs as well as user-centric needs).

The reaction plan built by the *Evolution Manager* may require suppressing and/or replacing parts or all of the Arch-based functional structure. This can involve re-using components from a *components storage,* and/or generating executable code on the fly by applying *transformations* on the models maintained by the *Models interpreters*. Executable code is one sort of digital models to represent a particular function. In current code-centric approaches, these are the only sorts of models considered at run time. As shown in the following section, they are not the only ones in a MDE approach to UI plasticity.

5.2 Models for UI plasticity

HCI design methods produce a large body of contemplative models such as scenarios, drawings, storyboards, and mock-ups. These models are useful reference material during the design process. On the other hand, because they are contemplative, they can only be transformed manually into productive models. Manual transformation supports creative inspiration, but is prone to wrong interpretation and loss of key information. Typically, contextual information elicited in a Contextual Design analysis, is lost during the development process. From explicit, contextual information becomes implicit within the executable program: the mapping between the real world contextual information and its corresponding translation into code does not exist. How can we improve the situation?

Given the current state of the art in HCI and SE, we see three possible realistic bridging points:

1- Moving from contemplative scenarios to productive use cases;
2- Bringing together productive task models with UML class diagrams to cover domain-dependent services and concepts;
3- Moving from the contemplative description of context of use into the terms of the ontological space presented in Section 2.1.1.

The framework presented in Figure 13 combines the second and third of our options with the MDE M1-M2 levels[9]. The M2 level is composed of a set of meta-models (and relations) that specify the structure of the "important" concepts of our problem space. The M1 level is populated with multiple models that each provides

[9] This framework is a revision of earlier reference frameworks from the authors [Calvary 01, Calvary 02] developed within the CAMELEON project (Context Aware Modelling for Enabling and Leveraging Effective interactiON) project: http://giove.isti.cnr.it/cameleon.html.

an "important" perspective on a given interactive system. According to the MDE principle, the M1-level models comply with the M2 level meta-models.

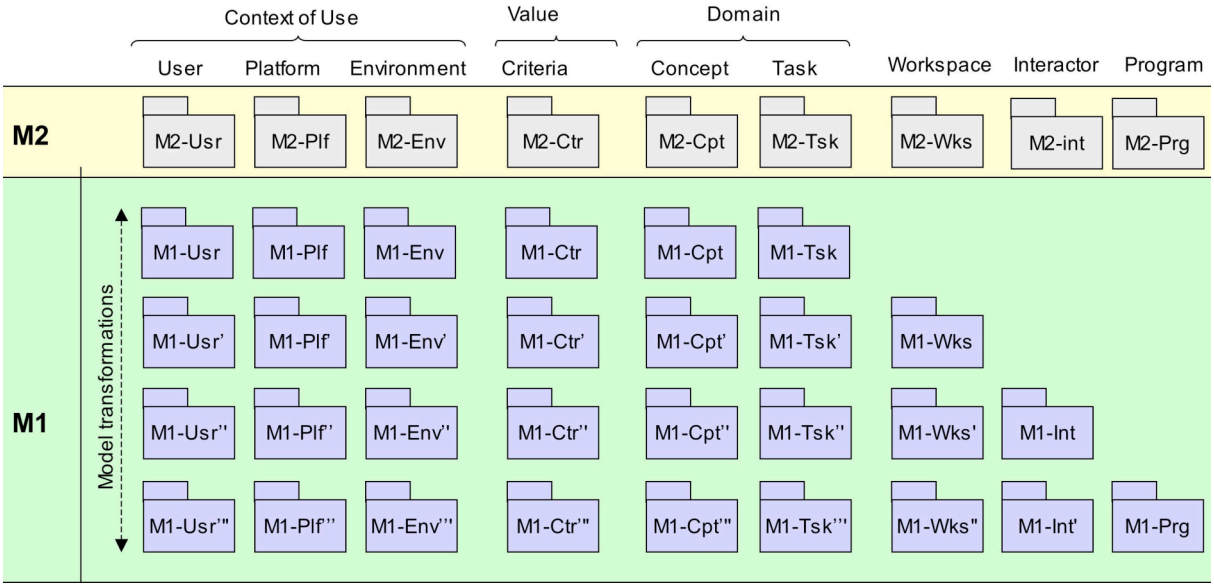


Figure 13. A Model Driven Engineering framework for plastic UI's. The set of columns represent different perspectives on a particular UI. In a column, all of the models *conform to* the same meta-model. A line corresponds to the result of transformation step. The framework makes explicit the need for revising models during the engineering process to fit in new constraints. For instance, an initial task model (M1-Tsk) might be tuned into another version (M1-Tsk') where a task interleaving operator (Figure 4a) is replaced with a sequence operator (Figure 4b) to cope with a small screen size.

Figure 14 makes explicit the definition of some of the M2 meta-models (and their relations) using UML as the meta-metamodel [Sottet 06]. Figure 16 shows examples of M1-level models for the home heating control system and their mapping with their respective meta-models.

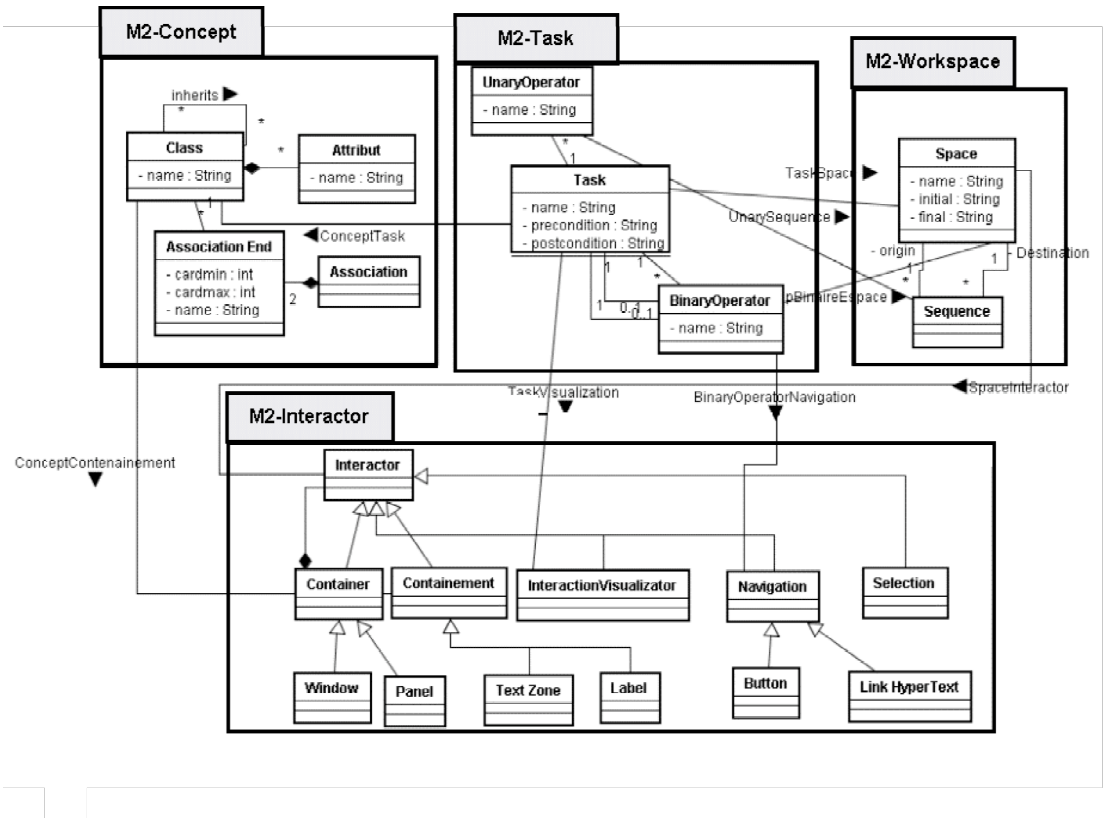


Figure 14. A subset of the meta-models of our framework for plastic UI's.

The M2-models (meta-models) include, but are not limited to the following:

M2-context of use defined as a specialization of the ontology presented in Section 2.1 where users, platforms and physical environments are sorts of Entities.

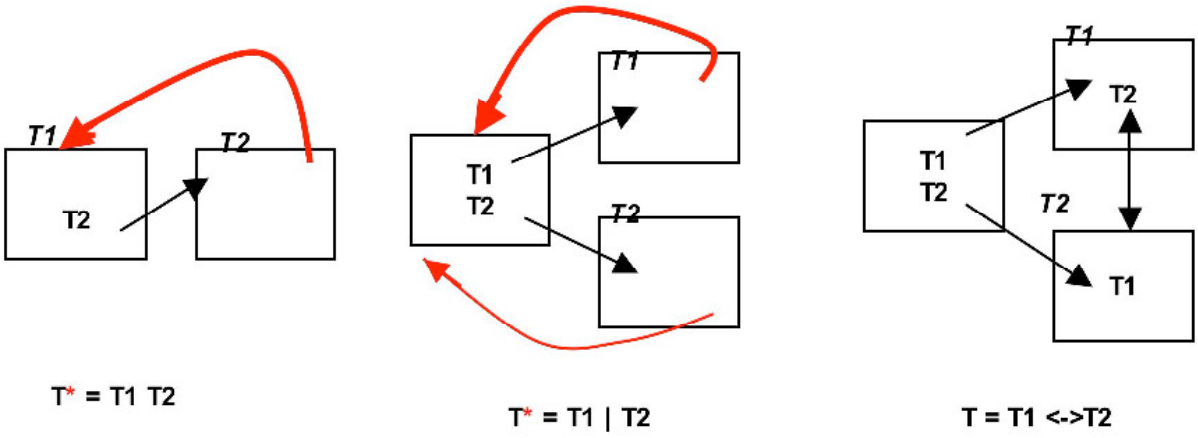
M2-evaluation criteria: the productive meta-model that results from a manual translation of the system value presented in Section 2.2.

M2-tasks and M2-concepts. As shown in Figure 14, a task has a name and pre- and post- conditions. It may be composed of other tasks by the way of a binary operator (such as the AND, OR, SEQ operators), or decorated with a unary operator (such as Optionality, Criticity, and Default option). BinaryOperators are related to navigation interactors to move between task spaces in the Concrete UI. A task manipulates concepts (denoted by the "ConceptTask" relation). In turn, a concept is a class composed of a set of attributes. This class may inherit from other classes, and may serve different roles. A concept is represented as interactor(s) in the Concrete UI by the way of the ConceptContainment relation. The TaskSpace relation shows how a task relates to a workspace.

M2-Workspace structures the task space at an abstract level, and M2-interactor describes the interactors that will populate workspaces in the Concrete UI. As shown by the definition of M2-Workspace, workspaces are chained with each other depending on the binary operator of the source tasks. A workspace is mapped into the Concrete UI as a container class interactor. This container interactor, which, in the GUI modality, may be a panel or a window, is populated with interactors that render concepts and binaryOperations (navigation operators).

M2-Program. A program is more than executable code. It covers multiple perspectives, typically: functional decomposition, mapping between functions and components, components configuration according to an architectural style[10], mapping between components and resources of the platform [Bass 98]. Typically, a component specifies the services (functions) it requires, the functions it provides, the quality of service it can support, the technical space it belongs to, the "code" it encapsulates. Components as defined in distributed systems are not "rich enough" to express HCI concerns. Typically, we would like to express that a component is able to support this set of tasks, that it can run both on a PC and a MacOS X platform, and that supports such and such evaluation criteria in such and such situation. The Comet is one attempt in this direction [Calvary 04].

M2-Transformations correspond to transformation descriptions that can be automated. Figure 15 show examples of typical transformations between source task models and target workspaces using the following rules: 1) there is one workspace per source task. 2) chaining between workspaces is based on task operators. Thus, "T= T1 T2" (which means: "to achieve T, T1 must be achieved first, followed by the achievement of T2") is transformed into two workspaces (shown in the picture as rectangles) that are accessible in sequence: the first workspace supports T1 and provides access to the workspace that supports T2. If T can be repeated, then the workspace that supports T2 must provide access to the workspace that supports T1. A more formal description of this type of transformation is presented next.



[10] An architecture style is a meta-model that includes a vocabulary of design elements (e.g., pipes and filters components), that imposes configuration constraints on these elements (e.g., pipes are mono-directional connectors between two filter components), and determines a semantic interpretation that gives meaning to the system description [Garlan, 1993].

Figure 15. Typical transformation patterns between task models (expressed with UAN operators) and workspaces.

In practice, how to exploit an M2-space?

5.3 Models in action

The M2-space offers many ways to produce plastic UIs. Let's see some examples.

The approach in conventional UI generation is to start from the production of a M1-task model and a M1-concept model, then to apply successive vertical transformations down to the generation of a M1-Program. In the example of Figure 16, the designer has produced the M1-Task model (bottom-right of the figure) and the M1-Concept model (top left) for the home heating control system. These M1 level models comply with their respective M2 level models. They have been transformed by vertical top down refinement into the Concrete UI shown at the top of Figure 16. This Concrete UI, as perceived and manipulated by the user, results from the interpretation of the corresponding M1-program by the model interpreters of Figure 11.

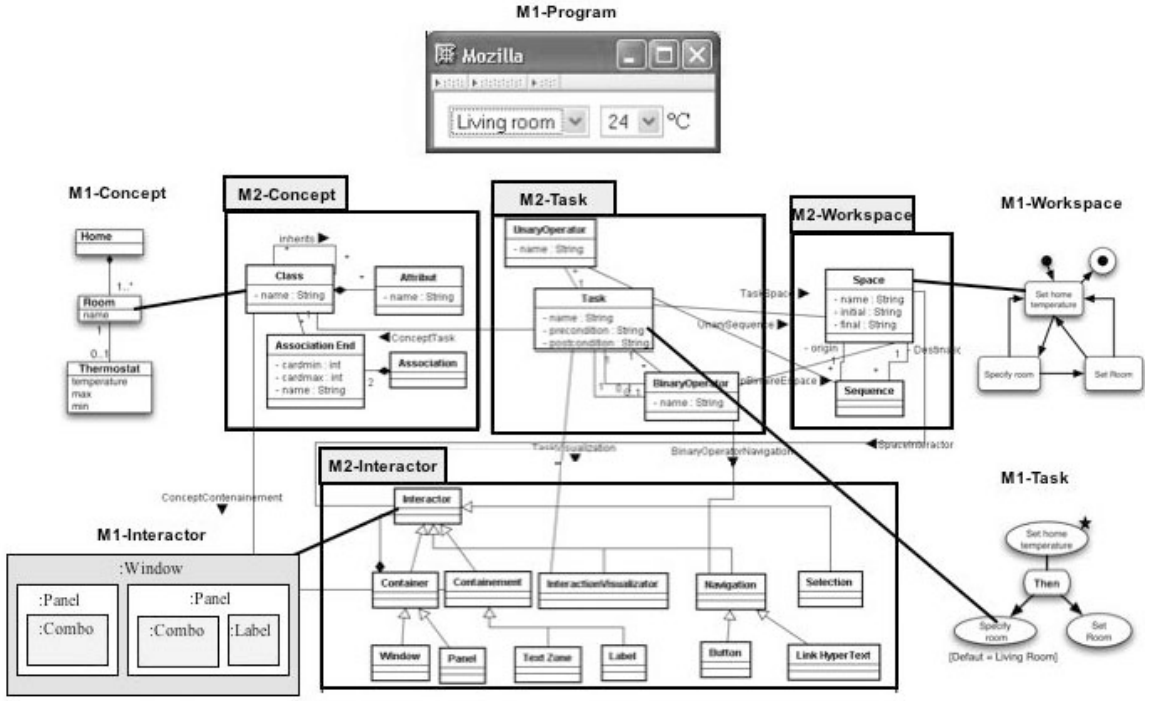


Figure 16. The home heating control system from an MDE perspective. A subset of the M1-level models that correspond to the situation c) of Figure 4, and their mapping with their respective M2-level models. (For simplification purpose, only a subset of the mappings have been represented.)

Because the M1 level models of our example are productive, the transformations can be performed automatically in the following way: the M1-Task model, which is comprised of one top level task and two subtasks, is transformed into one entry/exit workspace that leads to two workspaces (top right of Figure 16). Figure 17 shows the corresponding transformation description using ATL [Bézivin 03] as a transformation language.

Each workspace is then mapped into a container interactor (a window for the entry/exit workspace, and two panels within the window for the other two workspaces). The concepts of M1-Concept (room and thermostats) are mapped into their respective interactors within their container interactors: on one hand, a ComboBox for specifying the room; on the other hand, a ComboBox for setting the room temperature and a label to express the unit of measure (Celsius). The interactors are then transformed into executable code. This top-down approach is used in classic UI generation. However, in classic UI generation, there is no traceability between the M1 level models. Typically, once the code is generated, the task model is lost.

In our approach, all of the M1 level models maintain an explicit mapping between them. Since they are "alive" at run time, any model can be used on the fly to inform the adaptation process. Thus, our approach makes it possible to combine transformations at run time, mixing bottom-up transformations to reach the appropriate level of abstraction with horizontal transformations to switch, for example, to a distinct technological space, then

applying vertical refinement to produce an appropriate retargeted UI. Note that, because, mapping is maintained at a fine grain, transformations can be applied only to the portions of the UI that need to be adapted.

```
module M2TaskToM2Workspace {
  from M1Task : M2Task
  to   M1Workspace : M2Workspace
  -- One workspace per task
  rule TaskToSpace {
    from t : M2Task!Task
    to w : M2Workspace!Space (
      name <- t.name )
  }
  -- OrOperator to SequenceOperators
  rule OrOperatorToSequence{
    from o : M2Task!BinaryOperator (
      o.name = "or"
    )
    to motherToLeft : M2Workspace!Sequence (
      origin<- [TaskToSpace.w]o.motherTask,
      destination<-[TaskToSpace.w]o.leftTask)
```

Figure 17. An ATL transformation description based on the M2-space of Figures 14 and 16. The rule *TaskToSpace* creates one workspace *w* per source task *t* where *w* takes the name of *t*; The rule *OrOperatorToSequence* transforms all OR operators *o* between two tasks (*o.leftTask* and *o.rightTask*) into two sequence operators (from o.*motherTask* to *o.leftTask*, and *o.leftTask* to *o.rightTask*).

Other forms of optimization can be performed for known recurrent situations. For example, in CamNote and the Sedan-Bouillon Web site, two situations have been devised by design: the dynamic arrival and departure of a PDA with the dynamic migration of workspaces between the resources of the cluster. For these pre-planned situations, two types of control panels have been pre-computed and maintained in a components repository. They are dynamically retrieved as needed and mapped to the appropriate computing device.

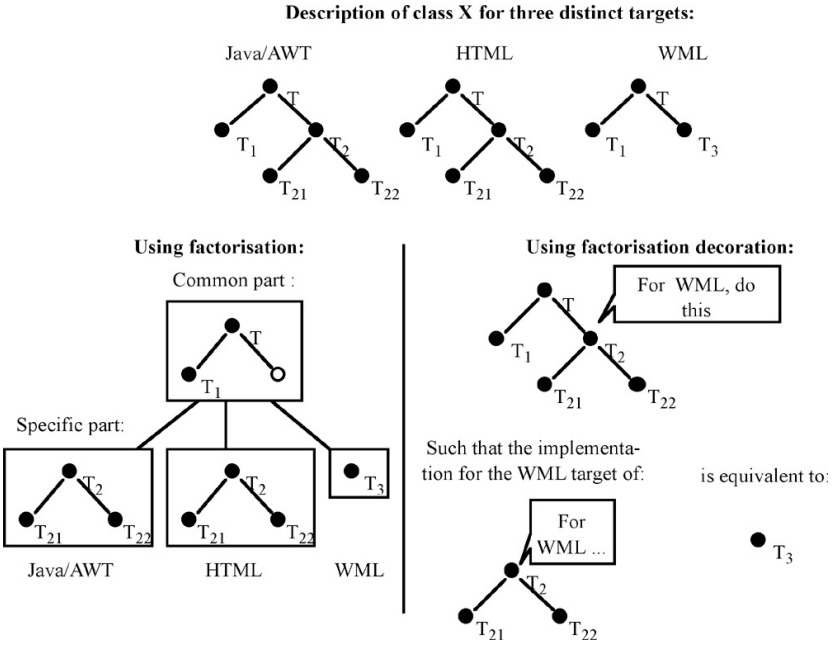


Figure 18. Factorisation applied to task models.

In an early version of the home heating control system, switching from a UI implemented in Java/AWT to a UI implemented in WML or HTML, was not performed on the fly: the target UIs were pre-computed according to the classic top-down approach using factorization and decoration transformations at every step of the vertical refinement process. Figure 18 illustrates the principle of these transformations. At the top, three M1-task models have been devised for the Java/AWT, HTML and WML-enabled target platforms. The task model obtained from

factorization is shown at the bottom left: 3 specific parts are linked to a shared part. The AVANTI's polymorphic tasks apply this method. On the bottom right, a unique task model is produced using decorations to express exceptions. Factorisation allows designers to focus on one target at a time, followed by the combination of the descriptions produced independently for each target into a single description where common parts are factored out. Decoration supports a different approach where designers focus on a reference target and then define exceptions.

5. Conclusion

Ad-hoc code-centric approaches to the development and deployment of plastic interactive systems are probably acceptable when the intended context of use is highly constrained. In ubiquitous computing, all aspects of user interface adaptation cannot be pre-programmed. Model Driven Engineering, which supports (re)computation from any level of abstraction, provides one possible avenue to address this new challenge.

In MDE, everything is a model. Models that represent a system do not float as independent things. Instead, they are organized into oriented graphs. Each graph represents an important principle of MDE: "compliance", "representation", and "transformations".

- Compliance enforces rigor. In MDE, the modeling world is organized into three levels of reasoning: the M1 model level, the M2 meta-level, and the M3 meta-meta level, where each level complies with the level above it.
- Representation supports the capacity to define multiple perspectives on a given thing.
- Transformation and mapping are fundamental to on the fly adaptation. It also provides the formal foundation for crossing the boundaries between technological spaces. This is where SE and HCI can cross-fertilize.

In HCI, Model-based approaches to the automatic generation of user interfaces have been investigated since the mid-eighties (Cf. the seminal COUSIN [Hayes 85] and Open-Dialogue [Schulert 85] User Interface Management Systems - UIMS's). Although promising, the UIMS technology has not found wide acceptance: for developers, it meant a new language to learn and designers felt severely limited by the constraints of stereotyped user interfaces [Myers 00]. The MDE-based framework we propose is intended to alleviate these problems: tools can be defined to encapsulate the low-level details of a particular language. Developers can "program" (produce models) in their favorite language since the result of this programming activity is a model that can be transformed into the appropriate target model(s). Creativity is not limited since, again, M0 level code can be vertically transformed into higher level of abstraction. The fundamental difference with the UIMS technology is that all models of a system are alive at run time: the run time system is not about executable code only.

Our MDE framework sets the global picture and principles for plastic user interfaces. It can be put in actions in many ways. Seminal work in plastic UI includes UIML [Abrams 94] and XIML [Puerta 01] that transform M1 level models into M0 level programs to support Logical Presentation level adaptation for centralized GUI. Tools for retargeting UI's such as Vaquita [Bouillon 02] and WebRevenge [Paganelli 03] correspond to a combination of bottom-up vertical, horizontal, and top-down vertical transformations. They lie within the same meta-meta level (the XML Technological Space), but they use distinct Level 2 meta-models. Vaquita and WebRevenge work off line. On the other hand, Digymes [Coninx 03] and Icrafter [Ponnekanti 01] generate Concrete User Interfaces (CUI) at run time where a renderer dynamically computes a CUI from a workspace level model expressed in XML. Websplitter [Perret 00] supports the distribution of web pages content at the interactor level across the interaction resources of heterogeneous clusters, but distribution is statically specified in an XML policy file. As proof of concepts, small size exemplars have been developed for different technological spaces. The challenge is to define appropriate M2-level meta-models. Transformation is also key to the success of this approach. TransformiXML of the UsiXML [Limbourg 04a, Limbourg 04b] meta-level environment, which is based on graphs transformations, is certainly a promising way. Thus, there is still a long way to go. Success will require a strong active collaboration between software engineering for HCI, software architecture, machine perception, and evaluation process.

6. Acknowledgements

This work has been supported by the Framework V FET Open GLOSS project (IST-2000-26070), CAMELEON (IST-2000-28323), and Framework VI Network of Excellence SIMILAR. The authors wish to thank Gaëtan Rey for the development of the contextor infrastructure, Lionel Balme and Alexandre Demeure for the implementation of CamNote and the Sedan-Bouillon Web site as well as for the development of the first version

of the run-time infrastructure for plastic UI's. Thanks to Jean-Marie Favre and Jean-Sébastien Sottet for their insights into MDE principles.

7. References

[Abrams 94] Abrams, M. Phanariou, C., Batongbacal, A., Williams, S. and Shuster, J. UIML: an appliance-independent XML User Interface Language. Proc. Of the 8th WWW conference, WWW'94.
[Arch 92] Arch, "A Metamodel for the Runtime Architecture of An Interactive System", The UIMS Developers Workshop, *SIGCHI Bulletin*, 24(1), ACM (1992).

[Balme 04] Balme, L., Demeure, A., Barralon, N., Coutaz, J., Calvary, G. CAMELEON-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces, Lecture Notes in Computer Science, Volume 3295 / 2004, Ambient Intelligence: Second European Symposium, EUSAI 2004, Markopoulos P., Eggen B., Aarts E. *et al.* (Eds), Springer-Verlag Heidelberg (Publisher), ISBN: 3-540-23721-6, Eindhoven, The Netherlands, November 8-11, 2004, pp 291-302.

[Bass 98] L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice*. Addison Wesley Publ., ISBN 0-201-19930-0 (1998).

[Berti 05] Berti, S. and Paternò F. Migratory multimodal interfaces in multidevice environments. In Proc. International Conference on Multimodal Interfaces (ICMI 05), ACM Publ., 2005, pp.92-99.

[Bézivin 01] Bézivin, J. In Search of a Basic Principle for Model Driven Engineering. Novatica Journal, Special Issue, March-April 2004.

[Bézivin 03] Bézivin, J., Dupé, G., Jouault, F., Pitette, G., Rougui., J. "First Experiments with the ATL Transformation Language: transforming XSLT into Xquery", in OOPSLA Workshop, Anaheim California USA, 2003.

[Bouillon 02] Bouillon, L., Vanderdonckt, J., Retargeting Web Pages to other Computing Platforms, Proceedings of IEEE 9th Working Conference on Reverse Engineering WCRE'2002 (Richmond, 29 October-1 November 2002), IEEE Computer Society Press, Los Alamitos, 2002, pp. 339-348.

[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.

[Calvary 02] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Souchon, N., Bouillon, L. Vanderdonckt, J. Plasticity of User Interfaces: A Revised Reference Framework, First International Workshop on Task Models and Diagrams for User Interface Design TAMODIA'2002, Bucarest, 18-19 July 2002, pp 127-134.

[Calvary 04] Calvary, G., Coutaz, J., Dâassi, O., Balme, L., Demeure, A. Towards a new generation of widgets for supporting software plasticity: the « comet », EHCI-DSVIS'2004, The 9th IFIP Working Conference on Engineering for Human-Computer Interaction Jointly with The 11th International Workshop on Design, Specification and Verification of Interactive Systems, Bastide, R., Palanque, P., Roth, J. (Eds), Lecture Notes in Computer Science 3425, Springer, ISSN 0302-9743, Hamburg, Germany, July 11-13, 2004, pp 306-323.

[Cockton 04] Cockton, G. From Quality in Use to Value in the World. In ACM Proc. CHI 2004, Late Breaking Results, April 2004, pp. 1287-1290.

[Cockton 05] Cockton, G. A development Framework for Value-Centred Design. In ACM Proc. CHI 2005, Late Breaking Results, April 2005, pp. 1292-1295.

[Coninx 03] Coninx K., Luyten K., Vandervelpen C., Van den Bergh J., Creemers B., Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems. In Proc. Mobile HCI, 2003.

[Coutaz 05] Coutaz, J., Crowley, J. , Dobson, S., Garlan, D. Context is Key, *Communications of the ACM*, ACM Publ., 48(3), March 2005, pp.49-53

Coutaz, J. Architectural Design for User Interfaces; The Encyclopedia of Software Engineering, J. Marciniak Ed., Wiley & Sons Publ., seconde édition, 2001.

[Elrad 01] Elrad, T., Filman, R., Bader, A. Aspect Oriented Programming. Special issue, Communication of the ACM, August 2001, 44(10).

[Favre 04a] Favre J.M., "Foundations of Model (Driven) (Reverse) Engineering", Dagsthul Seminar on Language Engineering for Model Driven Development, DROPS, http://drops.dagstuhl.de/portals/04101, 2004.

[Favre 04b] Favre J.M., "Foundations of the Meta-pyramids: Languages and Metamodels", DROPS, http://drops.dagstuhl.de/portals/04101, 2004.

[Hartson 90] Hartson, R., Siochi, A. & Hix, D. "The UAN: a user-oriented representation for direct manipulation interface designs". ACM Transaction on Information Systems (TOIS), 8(3), July 1990, pp. 181-203.

[Hayes 85] Hayes, P.J., Szekely, P., and Lerner, R.A. Design alternatives for user interface management systems based on experience with COUSIN. In Proc. Of the ACM Conf. on Human Factors in Computing Systems (CHI'85, San Francisco, CA, Apr. 14-18), 1985, pp. 169-175.

[Hinckley 03] Hinckley, K. Synchronous gestures for multiple persons and computers. Proc. 16th annual ACM symposium on User interface software and technology, UIST, 2003, pp. 149-158.

[Kurtev 02] I. Kurtev, J. Bézivin, M. Aksit, "Technological Spaces: an Initial Appraisal", CoopIS, DOA'2002 Federated Conferences, Industrial track, Irvine, 2002

[Limbourg 04a] Limbourg, Q. "Multi-path Development of User Interfaces", PhD of University of Louvain La Neuve, Belgium, 2004.

[Limbourg 04 b] Limbourg Q., Vanderdonckt J., Michotte B., Bouillon L., Lopez-Jaquero, V., "UsiXML: a Language Supporting Multi-Path Development of User Interfaces", Working Conference on Engineering for Human-Computer Interaction, 2004.

[Mens 04] Mens, T., Czarnecki, K. and Van Gorp, P. A taxonomy or Model Transformations. Dagstuhl Seminar Proc 04101. http://drops.dagstuhl.de/opus/volltexte/2005/11.

[Mori 02] Mori, G. Paternò, F. and Santoro, C. CTTE: Support for Developing and Analyzing Task Models for Interactive System Design. IEEE Transactions on Software Engineering, August 2002, pp. 797-813.

[Mori 04] Mori G., Paternò F., Santoro C. "Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions" IEEE Transactions on Software Engineering, August 2004.

[Myers 00] Myers B., Hudson S.E., Pausch R. "Past, Present, and Future of User Interface Software Tools", Transactions on Computer-Human Interaction (TOCHI), Vol 7, Issue 1, 2000.

[Myers 01] Myers, B. A., Using Handhelds and PCs Together, Communication of the ACM, Vol. 44, No 11, November 2001, pp. 34-41.

[Nielsen 93] Nielsen, J. Usability Engineering. London Academic Press. ISBN 0-12-518406-9, 1993.

[Paganelli 03] Paganelli, L., Paternò, F. A Tool for Creating Design Models from Web Site Code, International Journal of Software Engineering and Knowledge Engineering, World Scientific Publishing 13(2), pp. 169-189, 2003.

[Paternò 03] Paternò, F. ConcurTaskTrees: An Engineered Notation for Task Models, Chapter 24, in Diaper, D., Stanton, N. (Eds.), The Handbook of Task Analysis for Human-Computer Interaction, pp. 483-503, Lawrence Erlbaum Associates, 2003.

[Perret 00] Han R., Perret V., Naghshineh M., WebSplitter: A Unified XML Framework for Multi-Device Collaborative Web Browsing, Appeared in ACM Conference on Computer Supported Cooperative Work (CSCW) 2000.

[Planet] Planet MDE, "A Web Portal for the Model Driven Engineering Community" http://planetmde.org.

[Ponnekanti 01] Ponnekanti, S., Lee, B., Fox, A., Hanrahan, P., Winograd, T. Icrafter: a Service Framework for Ubiquitous Computing Environments. In *Proc. Ubicomp 2001*, G. Abowd, B. Brumitt, S. Shafer Eds., Springer Publ., LNCS 2201, 2001, pp. 57-75.

[Puerta 01] Puerta, A. Eisenstein. XIML: a common representation for interaction data. Proc. IUI01, ACM publ., 2001, pp. 214-215.

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

[Rey 05] Rey, G. Le Contexte en Interaction Homme-Machine ; le Contexteur. Thèse de doctorat Informatique préparée au Laboratoire de Communication Langagière et Interaction Personne-Système (CLIPS), Université Joseph Fourier, 1er Août, 2005

[Rosson 02] Rosson, M,B.,and Carroll, J. Usability Engineering, Scenario-Based Development of Human Computer InteractioN. Morgan Kaufmann Pub. 2201.

[Shackel 84] Shackel, B. The concept of usability. In Visual Display Terminals: Usability Issues and Health Concerns, J. Bennett et al. eds, Englewood Cliffs NJ: Prentice-Hall, ISBN 0-13-942482-2, 1984.

[Schulert 85] Schulert, A. J., Rogers, G. T., and Hamilton, J. A. ADM-A Dialogue Manager. In Proc. Of the ACM Conf. on Human Factors in Computing Systems (CHI'85, San Francisco, CA, Apr. 14-18), 1985, pp. 177-183.

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

[Streitz 99] Streitz, N., Geibler, J., Holmer, T., Konomi, S., Müller-Tomfelde, C., Reischl, W., Rexroth, P., Seitz, P., Steinmetz, R. i-LAND: An interactive Landscape for Creativity and Innovation. *In Proc. of the ACM conf. On Human Factors in Computer Human Interaction (CHI99)*, ACM, 1999, pp. 120-127.

[Sottet 06] Sottet, J.S., Calvary, G., Favre, J.M. Mapping Model: A First Step to Ensure Usability for sustaining User Interface Plasticity, MDDAUI'06 held in conjunction with MoDELS'06, Geneva, Italy, October 3, 2006

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

[Zizi 94] Zizi, M. and Beaudouin-Lafon, M. (1994) *Accessing Hyperdocuments through Interactive Dynamic Maps*. Proceeding of the European Conference on Hypertext Technology ECHT, September 19 – 23, 1994. Edinburgh, Scotland. ACM Press.