A Model-Driven Engineering Approach for the Usability of Plastic User Interfaces

Jean-Sébastien Sottet, Gaëlle Calvary, Joëlle Coutaz, Jean-Marie Favre

Université Joseph Fourier, 385 rue de la Bibliothèque, BP 53, 38041 Grenoble Cedex France
{Jean-Sebastien.Sottet, Gaelle.Calvary, Joelle.Coutaz, Jean-Marie.Favre}@imag.fr

Abstract. Ubiquitous computing has amplified the need for interactive systems to be able to adapt to their context of use (<User, Platform, Environment>) while preserving usability. This property is called *plasticity*. Until now, efforts have been put on the functional aspect of adaptation, neglecting the usability part of the definition. This paper investigates MDE mappings for embedding both the description and control of usability. A User Interface (UI) is a graph of models describing the UI from different perspectives ranging from user's tasks to the UI deployment on the context of use. Mappings link together these perspectives making explicit both the UI design rationale and the extent to which properties are preserved at runtime when the UI is transformed to target a new context of use. This paper provides a general definition and meta-model of *mapping* that are not devoted to Human-Computer Interaction (HCI). A mapping describes a transformation that preserves properties. A transformation is performed by a set of transformation functions that can be described either by a function and/or an execution trace. Mappings properties provide the designer with a means for both selecting the most appropriate transformation functions and previewing the resulting design. When applied to HCI, mappings are appropriate for both describing and controlling ergonomic criteria either at design time and/or at runtime.

Keywords: Adaptation, Context of use, Mapping, Meta-model, Model, Model transformation, Plasticity, Usability.

1 Introduction

In Human-Computer Interaction (HCI), plasticity refers to the ability of User Interfaces (UI) to withstand variations of context of use while preserving usability. Context of use refers to a set of observables that characterize the conditions in which a particular system is running. It covers three information spaces: the user model, the platform model, and the physical and social environment model. UI adaptation has been addressed using many approaches over the years, including Machine Learning [20], Model-Driven Engineering (MDE) [32], and Component-oriented services [29]. Regardless of the approach, the tendency has been to focus on functional aspects of adaptation. Usability has generally been regarded as a natural by-product of whatever

approach was being used. In this article, we propose to promote usability as a first class entity using a model-based approach.
The article is structured in the following way. Sections 2 and 3 motivate MDE for addressing plasticity and describe our principled approach. Section 4 introduces a running basic case study to serve as illustration. Sections 5 and 6 illustrate mappings on the case study and elaborate a general definition and meta-model of mapping that are then applied to HCI for supporting plasticity. Section 7 opens the work on many general and HCI perspectives.

2 Motivations for an MDE Approach

Although promising, the model-based approach to the development of UIs has not met wide acceptance: developers have to learn a new specification language, the connection between the specification and the resulting code is hard to understand and control, and the kinds of UI's that can be built are constrained by the underlying conventional toolkit [18]. However, this early work has established the foundations for transforming high-level specifications into executable code. In particular, the following steps now serve as references for designing and developing UIs: from the domain-dependent Concepts and Task models, an Abstract UI (AUI) is derived which in turn is transformed into a Concrete UI (CUI), followed by the Final UI (Figure 1) [35]. As discussed in [7], transformations can be combined and applied to any of these models to support UI adaptation. For example, VAQUITA [5] and WebRevEnge [22] reverse engineers HTML source files into more abstract descriptions (respectively AUI and task levels), and from there, depending on the tool, either retarget and generate the UI or are combined with retargeting and/or forward engineering tools (Figure 1). This means that developers can produce the models they are familiar with – including source code for fine-tuned elegant UIs, and then use the tools that support the appropriate transformations to retarget the UI to a different context of use. Transformations and models are at the heart of MDE.

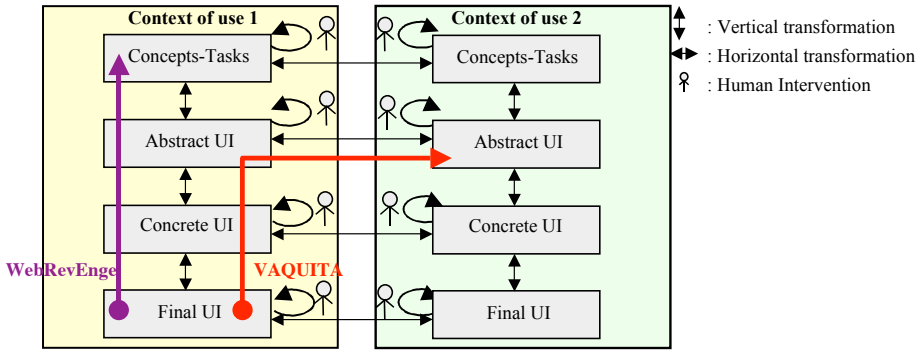


Fig. 1. A model-based framework [7] for characterizing tools.

The motivation for MDE is the integration of very different know-how and software techniques. Over the years, the field of software engineering has evolved into the

development of many 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*" [13]. 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, etc. In HCI, a java-based control panel running on a PDA can be used to control a web-based application running on a PC. 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. MDE aims at achieving integration by defining gateways between technological spaces. The hypothesis is that models, meta-models, models transformations and mappings, offer the appropriate means.

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 [4]. 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. Things and models are *systems*. Model is a *role of representation* that a system plays for another one. Models form oriented graphs (μ graphs) whose edges denote the μ relation "is represented by" (Figure 2). Models may be contemplative (they cannot be processed automatically by computers) or productive (they can be processed by computers). Typically, scenarios developed in HCI [26] are contemplative models of human experience in a specified setting. On the other hand, the task models exploited in TERESA [3] are productive.

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 TERESA [3] task model is compliant with CTT [24]. 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 (ε relation "is part of" on Figure 2), it is called a meta-model. CTT [24] is a meta-model for expressing specific task models.

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. In the same way, a *meta-meta-model 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. The OMG Model-Driven Architecture (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-meta-model. This top level is called M3, giving rise to meta-models, models and instances (respectively called M2, M1 and M0 levels). MDA is a specific

MDE deployment effort around industrial standards including MOF, UML, CWM, QVT, etc. 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 these issues.

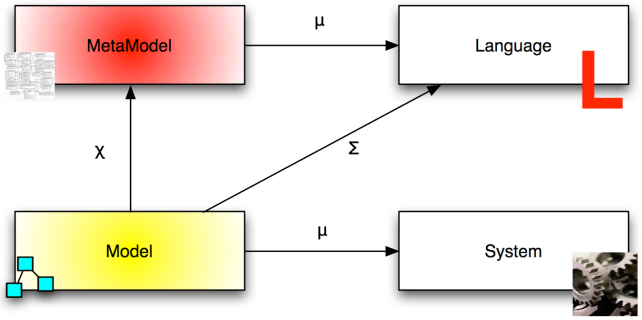


Fig. 2. Basic concepts and relations in MDE.

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 [15]. 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. 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 CUI model. Rendering is a transformation where tasks and their concepts are mapped into workspaces which, in turn, are mapped into windows populated with widgets in case of graphical UIs. The correspondence between the source task (and concepts) and its target workspace, window and widgets, is maintained as mappings. Mappings will be illustrated in Section 5 for the purpose of UI plasticity and generally meta-modeled in Section 6.

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, UI generators such as CTTE [17] produce UIs automatically from a task model. A transformation is *vertical* when the source and target models reside at different levels of abstraction (Figure 1). Traditional 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 (Figure 1). For example, translating a Java source code into C code preserves the original level of abstraction. Transformations are *endogenous* when the source and target models are expressed in the same language (i.e., are compliant to the same meta-model). Transformations are *exogenous* when sources and targets are expressed in different languages while belonging to the same technological space. When crossing technological spaces (e.g., transforming a Java source code into a JavaML document), then additional tools (called exporters and importers) are needed to bridge the gap between the spaces. *Inter-technological transformations* are key to knowledge and technical integration.

As discussed next, our approach to the problem of plastic UI is to fully exploit the MDE theoretic framework opening the way to the explicit expression of usability to drive the adaptation process.

3. MDE for UI Plasticity

Early work in the automatic generation of UIs [31] as well as more recent work in UI adaptation adhere only partially to the MDE principles. Our approach differs from previous work according to the following five principles.

Principle#1: An interactive system is a graph of M1-level models that expresses and maintains multiple perspectives on the system both at design time and runtime (Fig. 3). As opposed to previous work, an interactive system is not limited to a set of linked pieces of code. The models developed at design-time, which convey high-level design decision, are still available at runtime. A UI may include a task model, a concept model, a workspace (i.e. an AUI) model, and an interactor (i.e. a CUI) model linked by mappings. In turn, the UI components are mapped to items of the Functional Core of the interactive system, whereas the CUI elements (the interactors) are mapped to input and output (I/O) devices of the platform. Mappings between interactors and I/O devices support the explicit expression of centralized versus distributed UIs. The whole graph (Fig. 3) forms an ecosystem: a set of entities that interact to form an organized and self-regulated unit until some threshold is reached. When the threshold is reached, Principles #3 and #5 come into play.

Principle #2: *Transformations and mappings are models*. In the conventional model-driven approach to UI generation, transformation rules are diluted within the tool. Consequently, "the connection between specification and final result can be quite difficult to control and to understand" [18]. In our approach, transformations are promoted as models. As any model, they can be modified both at design-time and runtime at different degrees of automation. The same holds for mappings. In particular, mappings are decorated with properties to convey usability requirements. This aspect will be discussed in detail in Sections 5 and 6.

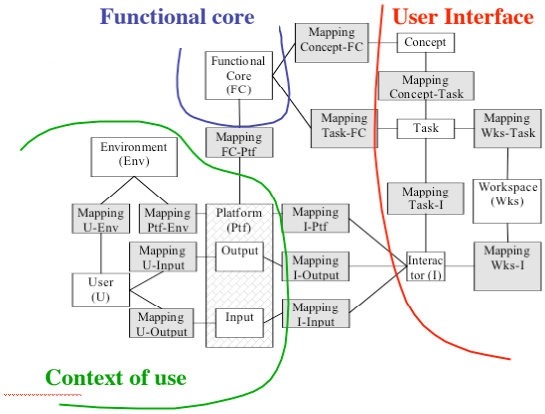


Fig. 3. A UI is a graph of models. Mappings define both the rationale of each UI element and the UI deployment on the functional core and the context of use.

Principle #3: *Design-time tools are runtime services*. The idea of creating UIs by dynamically linking software components was first proposed in the mid-eighties for the Andrew Toolkit [23], followed by OpenDoc, Active X, and Java Beans. However, these technical solutions suffer from three limitations: they are code centric, the assembly of components is specified by the programmer, and the components are supposed to belong to the same technological space. In our approach, any piece of code is "encapsulated" as a service. Some of them implement portions of the UI. We call them *UI services*. Others, the *UI transformers*, interpret the models that constitute the interactive system. In other words, the model interpreters used at design-time are also services at runtime. As a result, if no UI service can be found to satisfy a new context of use, a new one can be produced on the fly by UI transformers. In particular, the availability of a task model at runtime makes it possible to perform deep UI adaptation based on high-level abstractions.

Principle #4: *Humans are kept in the loop*. 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 to loss of key information. On the other hand, experience shows that automatic generation is limited to very conventional UIs. To address this problem, we accept to support a mix of *automated*, *semi-automated*, and *manually* performed transformations. Semi-automated and manual transformations may be performed by designers and/or end-users. For example, given our current level of knowledge, the transformation of a "value-centered model" [8] into a "usability model" such as that of [2], can only be performed manually by designers. Semi-automation allows designers (or end-users) to adjust the target models that result from transformations. For example, a designer may decide to map a subset of an AUI with UI services developed with the latest post-WIMP toolkit. The only constraint is that the hand-coded executable piece is modeled according to an explicit

meta-model and is encapsulated as a service. This service can then be dynamically retrieved and linked to the models of the interactive system by the way of mappings. With productive models at multiple levels of abstraction, the system can reason at run-time about its own design. In a nutshell, the components of a particular system at runtime can be a mix of generated and hand-coded highly tuned pieces of UI. By the way of a meta-UI [10], end-users can dynamically inform the adaptation process of their preferences.

Principle #5: *Close and open adaptiveness are complementary*. Designers cannot envision all of the contexts of use in which the future interactive system will be executed. As a result, there will be situations for which the system will not be able to adapt. To alleviate this problem, we suggest a mix of open and close adaptiveness. A system is *close-adaptive* when adaptation is self-contained. It supports the "innate" adjustments planned at the design stage as well as new adjustments produced by its own internal learning mechanisms. The system is *open-adaptive "if new adaptation plans can be introduced during runtime"* [21]. Applying the notion of close and open adaptiveness to UI plasticity, we say that an interactive system is *close-adaptive* for the contexts of use that fall within its *domain of plasticity* [6], that is the changes of contexts of use for which this system can adapt on its own. By design, an interactive system has an innate domain of plasticity. If it is able to learn adjustments for additional contexts of use, then the domain of plasticity extends dynamically, but this extension relies only on the internal computing capabilities of the system. In ubiquitous computing, unplanned contexts of use are unavoidable, forcing the system to go beyond its domain of plasticity. If continuity must be guaranteed, then the interactive system must call upon a tier infrastructure that takes over the adaptation process. The role of this infrastructure is to compute the best possible solution by applying new transformations and mappings on the system models and/or by looking for the appropriate services sitting somewhere in the global computing fabric. The interactive system, which relies on a tier infrastructure to perform adaptation, is *open-adaptive* for the contexts of use that do not fall within its domain of plasticity. The advantage of close-adaptation is that it is perfectly tuned to the interactive system for which it has been designed. Open adaptation may be less efficient, but has the potential to apply consistent adaptation principles to all of the interactive systems running in the interactive space. The balance between close and open adaptiveness is an open research issue.

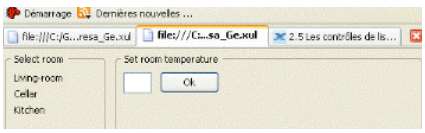
To summarize, our approach to the problem of UI plasticity brings together MDE (Model Driven Engineering) and SOA (Service Oriented Approach) within a unified framework that covers both the development stage and the runtime phase of interactive systems. In this paper, we investigate how usability can be described and controlled by the way of mappings given that an interactive system is a graph of models. We use a simple case study as an illustrative example before going into a more formal definition of the notion of mapping and its relation with that of transformation.

4. The Home Heating Control System: Overall Description

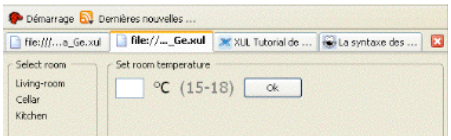
Our Home Heating Control System (HHCS) makes it possible for users to control the temperature of their home using different devices. Examples include a dedicated wall-mounted display, a Web browser running on a PDA, or a Java-enabled watch. As shown in Fig. 4, many variants of UIs are possible, depending on the device screen size, as well as the set of usability properties that HCI designers have elicited as key:

- From a functional perspective, the four UI's of Fig. 4 are equivalent: they support the same set of tasks, giving access to the same set of rooms (the living room, the cellar and the kitchen) where the room temperature may be set between 15°C and 18°C;
- From a non-functional perspective, these UI's do not satisfy the same set of usability properties. In particular, according to C. Bastien and D. Scapin's ergonomic framework [2], *prompting* (a factor for *guidance*), *prevention against errors* (a factor for *error management*), and *minimal actions* (a factor for *workload*) are not equally supported by the four UI solutions. In Fig. 4-a), the unit of measure (i.e. Celsius versus Fahrenheit) is not displayed. The same holds for the room temperature whose range of values is not made observable. As a result, prompting is not fully supported. In Fig. 4-b), the lack of prompting is repaired but the user is still not prevented from entering wrong values. Solutions in Fig. 4-c) and Fig. 4-d) satisfy both criteria. Moreover, Fig. 4-d) improves the *minimal actions* recommendation (a factor for *workload*) by eliminating the articulatory task consisting in selecting the room of interest. Other criteria such as *task compatibility* or *homogeneity-consistency* could be called upon as well to respectively praise the capacity of the UI to sustain the user's task, and the use of a same type of interactor (here links) to support the selection of a room (Fig. 4a to c).

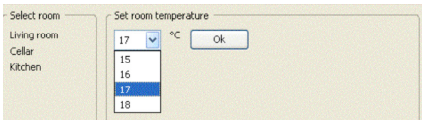
(a) The unit of measure and the valid temperature values are not observable



(b) The unit of measure and the valid temperature values are both observable



(c) The user is prevented from making errors



(d) The user is prevented from navigation tasks

Fig. 4. Four functionally-equivalent UIs that differ from the set of usability criteria used to produce them.

The purpose of this paper is to show how mappings are appropriate for conveying usability properties. Whatever the UI is (Fig.4-a, b, c or d), HHCS is a graph of models, each of them depicting a specific perspective. Each model (M1-level) is

compliant to a meta-model (M2-level). Fig. 5 shows a subset of the HHCS graph of models. The deployment on the functional core and the context of use is not depicted. Here, we use UML as a meta-meta-model (M3-level model).

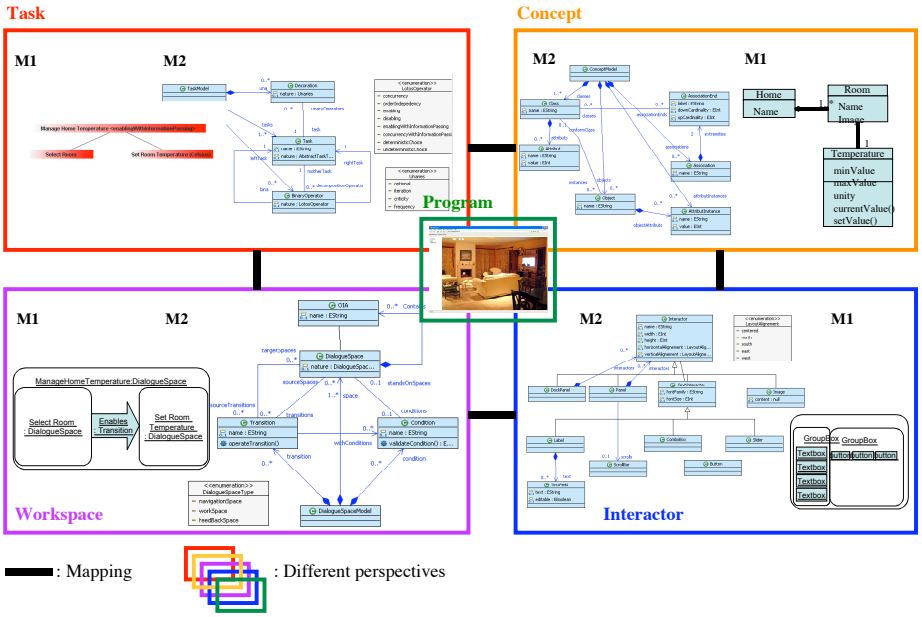


Fig.5 A subset of the graph of M1-models for HHCS. Each model is compliant to a meta-model (M2-level).

Fig. 6 shows a basic meta-UI for making observable and controllable a part of the graph of models to the user (either the designer and/or the end-user). In this first prototype, the meta-UI is limited to the task and platform models. By 'simply' selecting a task on the task model and selecting the platform(s) on which the user would appreciate to get the task, then the UI is re-computed and redistributed on the fly, thus ensuring UI consistency. On Fig. 6, two platforms are available (a PC HTML and a PC XUL). When the user deselects the PC XUL platform for the task "Select room" then the XUL UI is updated accordingly whilst the HTML UI is not changed. From an implementationnal perspective, the meta-UI is an OSGI service as well as the UI transformers (an embedded ATL transformation engine).

This first meta-UI demonstrates the existence of mappings only. In this first prototype, the properties of the mappings are neither observable nor controllable. This is the next implementationnal step for fully demonstrating the conceptual advances that are presented in the next sections. Section 5 is about the mappings in HHCS whilst section 6 makes a generalization providing a meta-model of mappings.

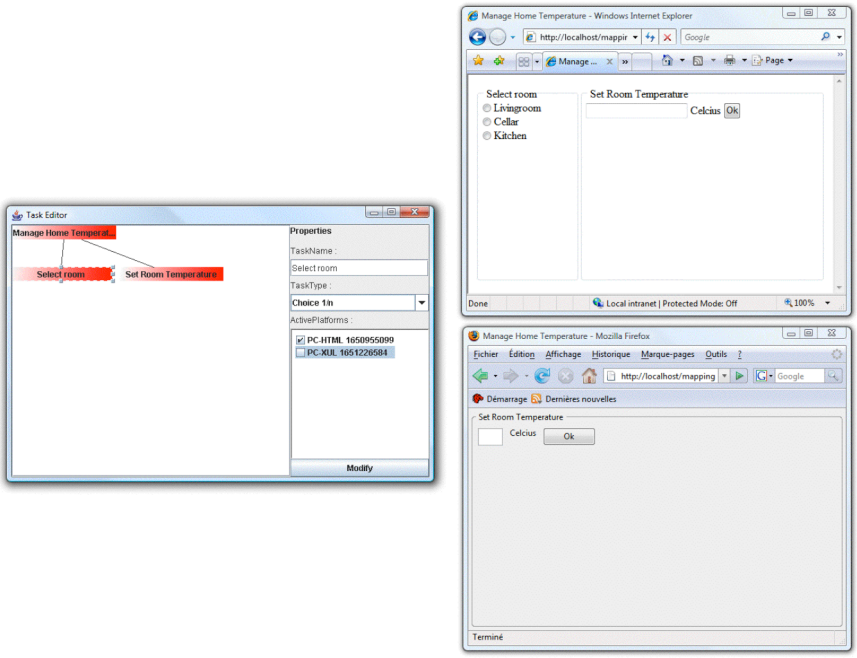


Fig.6 A basic meta-UI making it possible to the user to redistribute the UI by 'simply' manipulating the graph of models through the mappings between the tasks and the platform.

5. Mappings in HHCS

In the context of this work, we use Bastien and Scapin's framework as an example of a usability framework. For legibility, we limit our analysis to four of the eight criteria:

- *Task compatibility*;
- *Guidance* in its sub-criteria *Prompting* and *Grouping/Distinction of items*;
- *Error Management* in terms of *Error prevention*;
- *Workload* with regard to the *Minimal actions* it advocates.

Traditionally, criteria motivate the way abstract models are vertically transformed into more concrete models. Typically, the *Grouping/Distinction of items* (factor of *Guidance*) clearly motivates the structuration of the UI in terms of *workspaces* for *grouping* together the concepts that are manipulated in a same task and as a result making a *distinction* with the other tasks. Our approach promotes the elicitation of the usability properties on the mappings to support plasticity at runtime. Either the target element of the mapping is generated according to a transformation function that has been selected by the designer, or the link is made and described manually by the designer. For legibility, Fig. 7 focuses on three perspectives: the task model, the

concept model and the CUI. The AUI is not depicted. Workspaces are discussed at the CUI level, based on their representation.

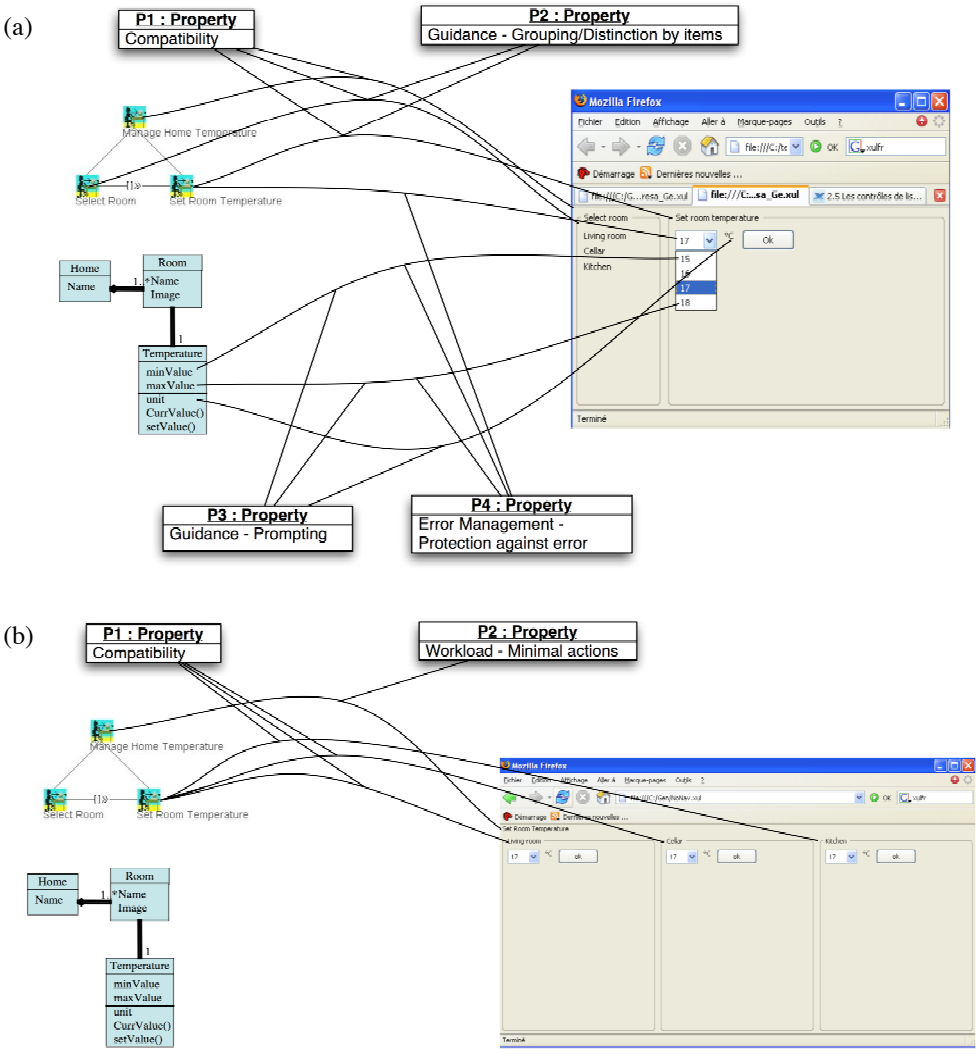


Fig. 7 Examples of mappings on the HHCS case study.

Fig. 7-a) makes explicit four properties. *Compatibility* (property P1) and *Grouping/Distinction of items* (property P2) deal with a set of mappings that link together tasks and workspaces. The UI fully supports the user's task and is well-structured. *Prompting* (property P3) and *Protection against error* (property P4) are related to the way concepts and tasks are represented in terms of interactors. Fig. 7-a) praises the fact that the temperature unit and the minimal/maximal values are

displayed, as a result favoring the prompting. This is not the case in Fig. 4-a). Prompting and drop lists prevent the user from errors.

Fig. 7-b) is limited to the mappings that discriminate the two UIs. In Fig. 7-b), the task "Select a room" becomes mental at the CUI level, as a result providing an incompatibility with the task model as is. On the other hand, it reduces the workload by eliminating the selection task. If workload is a key property, then the task model can be revised for ensuring consistency: either the "Select a room" task is transformed into a mental task, or it is 'distributed' on the "Set room temperature" task giving rise to "Set Kitchen temperature", "Set Living room temperature" and "Set Cellar temperature". Clearly, the loose of the concepts factorization suffers from non scalability. But above all, such transformations can damage the initial task model that has been produced by ergonomics. Keeping trace of all the transformations could help in making a separation of concerns between pure user-centered task models and others that have been tuned for engineering purpose.

In summary, mappings may be seen as rubber bands that link together different perspectives of a same UI. Their tensions reflect the extent to which usability is preserved. They break down when the UI goes outside its plasticity domain.

Next section presents our meta-model of mapping. This meta-model is general, applicable to HCI for reasoning on usability driven transformations.

6. Formal Definition of Mapping

In MDE, the term "mapping" is far from being clear. It is clearly related to the notion of transformation, but the distinction is not so clear. We first clarify the notion of transformation. Then, we propose a meta-model of mapping that involves this notion of transformation.

Figure 6 introduces three terms: *transformation model*, *transformation function* and *transformation instance*. They are illustrated on the mathematical domain. "f(x)=x+2" is a *transformation model* that is compliant to a mathematical meta-model. A transformation model describes (μ relation) a *transformation function* in a predictive way: here the set {(1,3),(2,4),(3,5)…} for "f" when applied to integers. A transformation function is the set of all the *transformation instances* inside the variation domain (here, the integers). A *transformation instance* is a subset (ε relation) of the *transformation function*. It is the execution trace of the function ("f").

In Fig. 8, the μ relation is refined into μp and μd. These relations respectively stand for *predictive* and *descriptive* representations. *Predictive* means that there is no ambiguity: the transformation model (e.g., "f(x)=x+2") fully specifies the transformation function. *Descriptive* refers to a qualifier (e.g., "growing"). This qualifier is not sufficient for specifying the transformation function, but it is a means for providing additional information. Fig. 8 illustrates two kinds of descriptive representations: one deals with a transformation model ("f(x)>x") whilst the other one deals with transformation instances ("growing"). In the first case, the description is made a priori whilst it is made a posteriori in the second case. A posteriori descriptions are subject to incompleteness and/or errors due to too few samples.

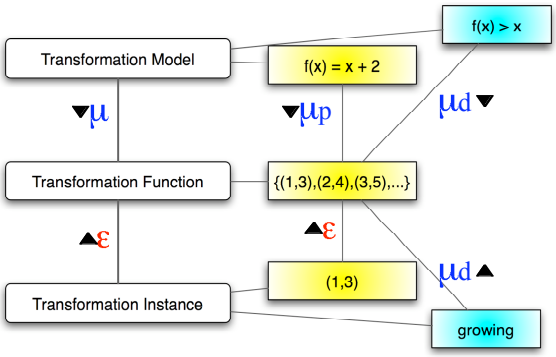


Fig. 8. Clarification of the notions of transformation model, transformation function and transformation instance.

Transformations are key for specifying mappings. The mapping meta-model provided in Fig. 9 is a general purpose mapping meta-model. The core entity is the *Mapping* class. A mapping links together entities that are compliant to *Meta-models* (e.g., Task and Interactor). A mapping can specify *Transformation functions* (e.g., {(Select the living room, Living room), (Select the kitchen, Kitchen), (Select the cellar, Cellar), …}) by patterns. A *Pattern* is a transformation model (e.g., (Select a concept, Concept), (Critical task, OK button), …). It links together source and target elements (*ModelElement*) to provide a predictive description of the transformation function. In addition, a mapping can describe the execution trace of the transformation function. The trace is made of a set of *Links* between *Instances* of *ModelElements*. The couple (Select the living room, Living room) is an example of *Link*.

A mapping conveys a set of *Properties* (e.g., "Guidance-Prompting"). A property is described according to a given reference framework (*Referential*) (e.g., Bastien&Scapin [2]). Because moving to an unfamiliar set of tools would impose a high threshold on HCI and software designers, we promote an open approach that consists in choosing the appropriate usability framework, then generating and evaluating UIs according to this framework. General frameworks are available such as Shackel [28], Abowd et al., [1], Dix et al. [11], Nielsen [19], Preece [25], IFIP Properties [12], Schneiderman [30], Constantine and Lockwood [9], Van Welie et al. [38], as well as Seffah et al. [27] who propose QUIM, a unifying roadmap to reconcile existing frameworks. More specific frameworks are proposed for web engineering (Montero et al. [16]), or for specific domains (for instance, military applications). Closely related to UI plasticity, Lopez-Jacquero et al.'s propose a refinement of Bastien and Scapin's framework, as a usability guide for UI adaptation [14]. Whatever the framework is, the properties are descriptive. They qualify either the global set of mappings or one specific element: a mapping, a pattern or a link. Examples are provided in Fig. 7.

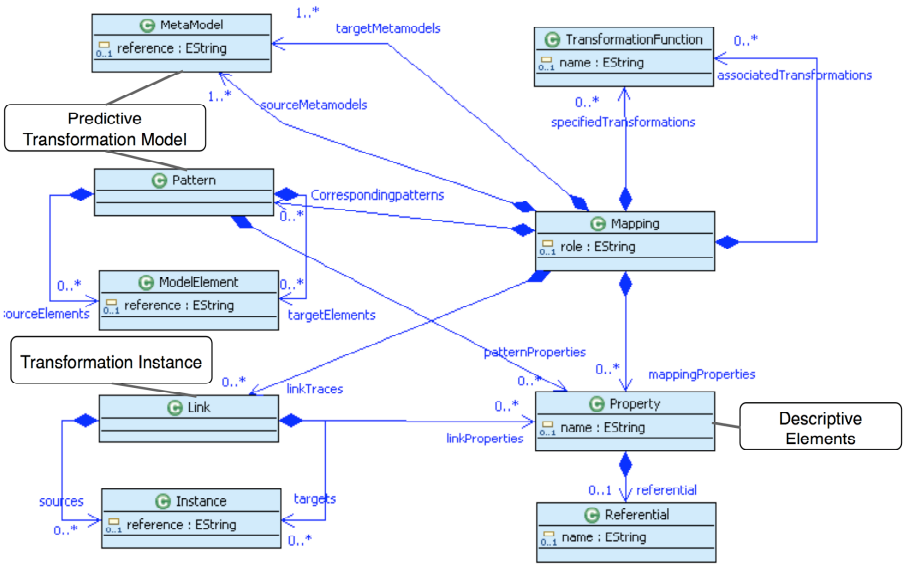


Fig. 9. A mapping meta-model for general purpose. The composition between Mapping and Metamodel is due to the Eclipse Modeling Framework.

Associated transformations (see the UML association between the classes Mapping and TransformationFunction) are in charge of maintaining the consistency of the graph of models by propagating modifications that have an impact on other elements. For instance, if replacing an interactor with another one decreases the UI homogeneity-consistency (one of the eight Bastien and Scapin's criteria), then the same substitution should be applied to the other interactors of the same type. This is the job of the associated functions which perform this adaptation locally.

Our mapping meta-model is general. The HCI specificities come from the nature of the meta-models and the properties elicited in the framework. Currently in HCI, effort is put on UI meta-modeling (see UsiXML [37] for instance) but the mapping meta-model remains a key issue. Are the traditional usability frameworks still appropriate for plasticity? Should new criteria (e.g., continuity [36]) be added? Which metrics could be incorporated in order to make it possible for the system to self-evaluate? Next section elaborates on perspectives for both HCI and MDE communities.

7 Conclusion and perspectives

In 2000, B. Myers stated that model-based approaches had not found a wide acceptance in HCI. They were traditionally used for automatic generation and appeared as disappointing because of a too poor quality of the produced UIs. He envisioned a second life for models in HCI empowered by the need of device

independence. In our work, we promote the use, the description and the capitalization of elementary transformations that target a specific issue.

A UI is described as a graph of models and mappings both at design time and runtime. At design time, mappings convey properties that help the designer in selecting the most appropriate transformation functions. Either the target element of the mapping is generated according to the transformation function that has been selected, or the link is made by the designer who then describes the mapping using a transformation model. We envision adviser tools for making the designer aware of the properties he/she is satisfying or neglecting.

At runtime, mappings are key for reasoning on usability. However, it is not easy as (1) there is not a unique consensual reference framework; (2) ergonomic criteria may be inconsistent and, as a result, require difficult tradeoffs. Thus, (1) the meta-model will have to be refined according to these frameworks; (2) a meta-UI (i.e., the UI of the adaptation process) may be relevant for negotiating tradeoffs with the end-user.

Beyond HCI, this work provides a general contribution to MDE. It defines a mapping meta-model and clarifies the notions of mapping and transformation. Mappings are more than a simple traceability link. They can be either predictive (transformation specifications) or descriptive (the properties that are conveyed), as a result covering both the automatic generation and the hand-made linking. Moreover mapping models can embed transformation in order to manage models consistency. This is new in MDE as most of the approaches currently focus on direct transformation. Our mapping meta-model will be stored in the international Zoo of meta-models: the ZOOOMM project [39].

Acknowledgments. This work has been supported by the network of excellence SIMILAR and the ITEA EMODE project. The authors warmly thank Xavier Alvaro for the implementation of the prototype.

References

1. Abowd, G.D., Coutaz, J., Nigay, L. Structuring the Space of Interactive System Properties, Engineering for Human-Computer Interaction, Larson J. & Unger C. (eds), Elsevier, Science Publishers B.V. (North-Holland), IFIP, 1992, pp 113-126
2. Bastien, J.M.C, Scapin, D. Ergonomic Criteria for the Evaluation of Human-Computer, Technical report INRIA, N°156, June 1993
3. Berti, S., Correani, F., Mori, G., Paterno, F., Santoro, C. Conference on Human Factors in Computing Systems, CHI '04 extended abstracts on Human factors in computing systems, Vienna, Austria, 2004, pp 793-794
4. Bézivin, J. In Search of a Basic Principle for Model Driven Engineering, CEPIS, UPGRADE, The European Journal for the Informatics Professional V(2):21—24, 2004
5. Bouillon, L., Vanderdonckt, J. Retargeting of Web Pages to Other Computing Platforms with VAQUITA, Proceedings of the Ninth Working Conference on Reverse Engineering (WCRE'02), 2002, pp 339
6. Calvary, G., Coutaz, J., Daassi, O., Balme, L., Demeure, A. Towards a new generation of widgets for supporting software plasticity: the 'comet' EHCI-DSVIS'04, Springer, Hamburg, Germany, 11-13 June 2004, pp 306-323

7. Calvary, G., Coutaz J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J. A unifying reference framework for multi-target user interfaces, Interacting With Computers, Vol. 15/3, 2003, pp 289-308
8. Cockton, G. A development Framework for Value-Centred Design, In ACM Proc. CHI 2005, Late Breaking Results, 2005, pp 1292-1295
9. Constantine, L.L., Lockwood, L.A.D. Software for Use: A Practical Guide to the Models and Methods of Usage-Centred Design, New-York, Addison-Wesley, 1999
10. Coutaz, J. Meta-User Interfaces for Ambient Spaces, Interational Workshop on Task Model and Diagram (TAMODIA'06), Bucarest, Romania, 2006
11. Dix, A., Finlay, J., Abowd, G., Beale, R. Human-Computer Interaction, Prentice-Hall, New-Jersey, 1993
12. IFIP Design Principles for Interactive Software, IFIP WG 2.7 (13.4), C. Gram & G. Cockton Eds., Chapman&Hall Publ., 1996
13. Kurtev, I., Bézivin, J., Aksit, M. Technological Spaces: An Initial Appraisal, In Proceedings of the Confederated International CoopIS, DOA, and ODBASE 2002, Industrial track, Irvine, CA, USA, 2002
14. Lopez-Jaquero, V., Montero, F., Molina, J.P., Gonzalez, P. A Seamless Development Process of Adaptive User Interfaces Explicitly Based on Usability Properties, EHCI'04, pp 289-291
15. Mens T., Czarnecki K., Van Gorp P. A Taxonomy of Model Transformations Language Engineering for Model-Driven Software Development, Dagstuhl February - March 2004
16. Montero, F., Vanderdonckt, J., Lozano, M. Quality Models for Automated Evaluation of Web Sites Usability and Accessibility, In proc. of the International Conference on Web Engineering, ICWE'2004, July 28-30, Munich, 2004
17. Mori, G., Paternò, F., Santoro, C. CTTE: Support for Developing and Analyzing Task Models for Interactive System Design, IEEE Transactions on Software Engineering August 2002, pp 797-813
18. 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
19. Nielsen, J. Heuristic evaluation, In Nielsen, J., and Mack, R.L. (Eds.), Usability Inspection Methods, John Wiley & Sons, New York, NY, 1994
20. Njike, H., Artières, T., Gallinari, P., Blanchard, J., Letellier, G. Automatic learning of domain model for personalized hypermedia applications, International Joint Conference on Artificial Intelligence, IJCA, Edinburg, Scotland, 2005, pp 1624
21. Oreizy, P., et al. An Architecture-based Approach to Self-Adaptive Software. IEEE Intelligent Systems, May-June, 1999, pp 54-62
22. Paganelli, L., Paternò, F. Automatic Reconstruction of the Underlying Interaction Design of Web Applications, Proceedings Fourteenth International Conference on Software Engineering and Knowledge Engineering, ACM Press, Ischia, July 2002, pp 439-445
23. Palay, A., Hansen, W. Kazar, M., Sherman, M., Wadlow, M., Neuendorffer, T., Stern, Z., Bader, M., Peters, T. The Andrew Toolkit: An Overview, In Proc. On Winter 1988 USENIX Technical Conf., USENIX Ass., Berkeley, CA, 1988, pp 9-21
24. Paterno', F., Mancini, C., Meniconi, S. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models, Proceedings Interact'97, July'97, Sydney, Chapman&Hall pp 362-369
25. Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., Carey, T. Human-Computer Interaction, Wokingham, UK, Addison Wesley Publ., 1994
26. Rosson, M.B., Carroll, J.M. Usability Engineering: Scenario-Based Development of Human-Computer Interaction, San Francisco, CA: Morgan Kaufman, 2002
27. Seffah, A., Donyaee, M., Kline, R. B. Usability and quality in use measurement and metrics: An integrative model. Software Quality Journal, 2004

28. Shackel, B. Usability-Context, Framework, Design and Evaluation. In Human Factors for Informatics Usability, Cambridge University Press, 1991, pp 21-38
29. Sheshagiri, M., Sadeh, N., Gandon, F. Using Semantic Web Services for Context-Aware Mobile Applications, In Proceedings of ACM MobiSys2004 Workshop on Context Awareness, Boston, Massachusetts, USA, June 2004
30. Shneiderman, B. Designing User Interface Strategies for effective Human-Computer Interaction (3rd ed), Addison-Wesley Publ., 1997, 600 pages
31. da Silva, P. User Interface Declarative Models and Development Environments: A Survey Proceedings of DSV-IS2000, 1946- Springer, Limerick, Ireland, June 5-6, 2000, pp 207-226
32. Sottet, J.S., Calvary, G., Favre, J.M., Coutaz, J., Demeure, A., Balme, L. Towards Model-Driven Engineering of Plastic User Interfaces, in Conference on Model Driven Engineering Languages and Systems (MoDELS'05) satellite proceedings, Springer LNCS, 2005, pp 191-200
33. Sottet, J.S., Calvary, G., Favre, J.M. Towards Mappings and Models Transformations for Consistency of Plastic User Interfaces, The Many Faces of Consistency, Workshop CHI2006, Montréal, Québec, Canada, April 22-23, 2006
34. Sottet, J.S., Calvary, G., Favre, J.M. Mapping Model: A First Step to Ensure Usability for sustaining User Interface Plasticity, Proceedings of the MoDELS'06 Workshop on Model Driven Development of Advanced User Interfaces, October 3th 2006
35. Thevenin, D. Plasticity of User Interfaces: Framework and Research Agenda, In Proc. Interact99, Edinburgh, , A. Sasse & C. Johnson Eds, IFIP IOS Press Publ., pp 110-117
36. Trevisan, D., Vanderdonckt, J., Macq, B. Continuity as a usability property, HCI 2003 - 10th Intl Conference on Human-Computer Interaction, Heraklion, Greece, June 22-27, 2003, Vol. I, pp 1268-1272
37. UsiXML, http://www.usixml.org/
38. Van Welie, M., van der Veer, G.C., Eliëns, A. Usability Properties in Dialog Models: In 6th International Eurographics Workshop on Design Specification and Verification of Interactive Systems DSV-IS'99, Braga, Portugal, 2-4 June 1999, pp 238-253
39. Zooomm Project: http://www.zooomm.org