Journal on Multimodal User Interfaces, Vol. 1, No. 2, 2007

A LANGUAGE PERSPECTIVE ON THE DEVELOPMENT OF PLASTIC MULTIMODAL USER INTERFACES

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

Laboratoire LIG, Université Joseph Fourier, Grenoble, France
{jean-sebastien.sottet; gaelle.calvary; joelle.coutaz; jean-marie.favre}@imag.fr

Jean Vanderdonckt, Adrian Stanciulescu

Louvain School of Management (LSM), Université catholique de Louvain, Louvain-la-Neuve, Belgium
{jean.vanderdonckt; adrian.stanciulescu}@uclouvain.be

Sophie Lepreux

Université de Valenciennes et du Hainaut-Cambrésis, Valenciennes, France
sophie.lepreux@univ-valenciennes.fr

ABSTRACT

Designing complex interactive systems requires the collaboration of actors with very different background. As a result, several languages and tools are used in a single project with no hope for interoperability. In this article, we examine whether a universal language is a realistic approach to UI specification by looking for answers into the domain of Linguistics while finding analogies in software engineering. Then, we explore one particular avenue from main-stream software engineering: that of Model Driven Engineering where the notion of transformation is key to the definition of bridges between languages and tools. Building upon these two analyses, we then show how model-driven engineering can be successfully exploited in the development and execution of plastic multimodal UIs illustrated with a variety of complementary tools.

KEYWORDS

HCI – User Interface – Interoperability – Metamodel – Model Driven Engineering

1. INTRODUCTION

Many scenarios for ambient computing praise the power that will result from the interaction between "mixed-reality" and "embodied-virtuality" [1]. End-users will be able to construct new personalized mixed objects by assembling digital entities with physical objects [2]. As shown in Fig. 1, they will dynamically assign roles to everyday objects using natural, reality-based knowledge and skills [3]. They will distribute portions of User Interfaces (UIs) across interaction resources of many sorts (walls, tables, ear plugs) whose availability will vary dynamically as users move in an augmented world [4]. As demonstrated by the increasing success of mash-ups, new services will be obtained from the fusion of multiple data sources.

However, with this power arise new requirements. Among these requirements is the capacity for UIs to be plastic [5], that is, to be able to adapt dynamically to the context of use (e.g., to users, platforms, as well as to social and physical environments). In the future, this adaptation will be radically polymorphic along a continuous scale, from centralized extra-small

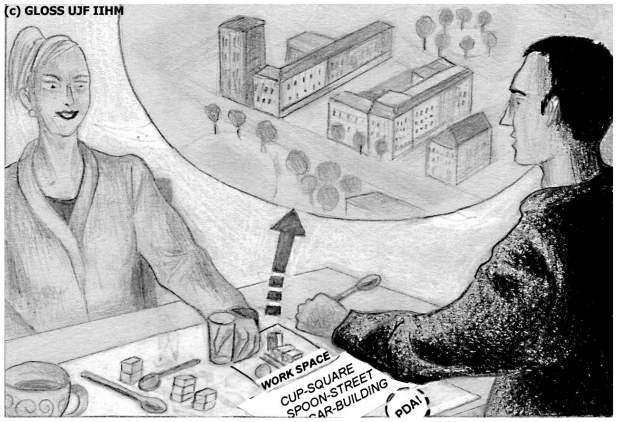


Figure 1: *Town architects, Bob and Jane, assigning roles to everyday objects as they are discussing the design of a town informally in a Café. Here, lump of sugars and spoons are used to represent buildings and streets respectively.*

graphical UIs for motes, to distributed, post-WIMP, multimodal room-size interaction styles including "emo-robots" that will recognize and evoke human emotions. To make this possible, a unifying framework for next-generation UIs is needed. There have been some attempts in this direction including Ullmer and Ishii's framework on tangible UIs [6], Fishkin's et al. proposal on embodied interfaces [7], or Nigay's and Coutrix model for mixed reality [8]. These contributions provide designers with helpful analytic tools but they concentrate on classes of new UIs rather than on the software aspects that support the morphing between these classes.

In this paper, we concentrate on the problem of UI adaptation to the diversity of UI classes under dynamic and unpredictable constraints. Our long-term goal is to provide software designers, as well as end-users and the system itself, with the appropriate tools and mechanisms to design, develop, and configure plastic UIs. Our approach to the problem of UI plasticity is based on the following two observations: (1) Not all situations can be envisioned at design time. (2) A plethora of tools and lan-

guages for designing and developing UIs have been created with little care for interoperability.

- Observation 1: Not all situations can be envisioned at design time. If contexts of use can be defined at design time, then UI development can be structured as a step-by-step process, from UI specifications to executable code. In this case, the system can self-adapt to reference contexts of use. On the other hand, if the run-time constraints are dynamic and unpredictable, then it becomes necessary for the system to reason about its own design to adapt to situations that have not be considered by the designers. This new requirement advocates making the design rationale available at run time (unless we accept that system adaptation is limited to low-level cosmetic adjustments). This means that the models produced during the development process should be exploitable at run time by the system as well as by end-users [9].
- Observation 2: A plethora of tools and languages for designing and developing UIs have been created with little care for interoperability. To design and develop UIs, practitioners have available a wide variety of specialized tools and languages. For example, both CTT [10] and KMAD [11] cover the specification of task models, but they do not interoperate. The choice between these options depends primarily on the designer's background (skills and habits), rarely on objective criteria. In order to support a "low threshold" in learning how to use a new tool [12], designers and developers should be able to use the tools with which they are familiar (unless "high ceiling" can be expected about how much can be done with the tool). Unfortunately, this constraint, which leads to the production of models that do not interoperate, impedes their cooperation at run time.

The conflicting requirements from observations 1 and 2 may be solved in two ways: either a universal language is defined to cover all aspects of UI specifications, or we accept the diversity of tools and languages but we find a way to define bridges between them. This article explores these two options. First, we examine whether a universal language is a realistic approach to UI specification by looking for answers into the domain of Linguistics while finding analogies in software engineering. Then, we explore one particular avenue from main-stream software engineering: that of Model Driven Engineering (MDE) where the notion of transformation is key to the definition of bridges between languages. Building upon these two analyses, we then show how MDE can be successfully exploited in the development and execution of plastic multimodal UIs illustrated with a variety of complementary tools.

2. LINGUISTICS AND SOFTWARE

Linguistics is the scientific discipline of the study of languages [13]. Among other things, it provides answers to the following two questions: "what are the functions of languages?" and "why are there so many languages?"

2.1. The functions of languages

According to linguistics, the two most important functions of a language is to enable us (1) to reason about things, and (2) to communicate with each other [14][13]. Software languages have the same functions except that they can be interpreted by computers, not by human actors only.

How humans think about something. Written or graphical languages help us in reasoning about complex problems. "Thinking aloud" about an interface design or drawing a task diagram on a napkin corresponds to the same function of a language: thinking or reasoning about a UI to be built. Note that this scenario does not require the language to be well defined: the actor "understands himself" anyway.

How humans communicate. As communication involves multiple actors, there is a strong need for an explicit language description or at least some common level of agreement between the actors. In the simplest case, the actors speak the same language. This is the situation where two Human-Computer Interaction (HCI) designers communicate to merge their task diagrams. But there are situations where communication involves actors who speak different languages or dialects. Various scenarios are possible. The first one is to use a vehicular language to communicate. In Europe, this role has been played by Latin and French. Unified Modeling Language (UML) plays a similar role in the software engineering landscape when software engineers use the core of UML (i.e. a subset of class diagrams). Some actors may use their own dialects within their own community (also known as vernacular languages), and switch to the vehicular language when going outside. Diglossy is the ability for an actor to switch from one language to another depending on the situation: a requirements engineer may elaborate use case diagrams when speaking with clients, and then switch to task diagrams when talking to HCI designers. Summing up, humans need to communicate with each other, and this implies the existence of bridges between existing languages and communities.

2.2. The Multiplicity of Languages

There are, at least, three reasons for having so many languages and dialects: languages are specific to communities, language is power, and a language has a purpose.

Languages and Communities. According to Linguistics, "languages are the cements of our societies" [13]. There are many languages because there are many communities, and there are many communities because there are many languages. Speaking a language is what makes the difference between those that are in the community, and the "foreigners", that are not. In ancient Greece, Barbarian was the name given to all people who did not speak Greek. Similarly, in the software world, software languages can lead to a strong feeling of identity as people can define themselves as C++ programmers, Haskell programmers, UML modelers, and so on.

Language is Power. Communities are always fighting to extend their power and have greater influence over other communities. Imposing a language over another one is a way to remove the boundaries between two communities and integrate the dominated culture. The war of languages is a well-established concept in Linguistics [14] [13]. It is important to recognize that the development of languages is not linked to their intrinsic properties but to external and political factors. In the software world too, the most used software languages are not necessarily the "best" ones (according to a given set of requirements). Language expansion always requires some resources and efforts. For example, Ada has benefited from the support of governmental institutions in America and Europe. It has been designed by the USA' Department of Defense (DoD) to take over the 500 different languages used at that time by the DoD. In the context of modeling languages, Object Management Group (OMG) has done a similar job in promoting UML. The universal language is however a recurrent myth and the history of languages has always been based on converging as well as on divergent processes.

Languages and Purposes. As seen above, many factors explain the babelization of the software landscape, but separation

of concerns is one of the key aspects. For many years, computer science has dealt only with programming languages. Since then, many other kinds of languages have emerged: specification languages (e.g. Z, VDM), modeling languages (e.g. SADT, UML), architectural description languages (e.g. Wright, ACME), and so on. This fragmentation of software languages helps in dealing with different levels of abstraction. In principles, they are independent from application domains, although an application-driven approach is emerging with the notion of Domain Specific Languages (DSL). In the area of Linguistics this corresponds to Jargons, Special Purpose Languages and Restricted Languages [13].

DSLs are expected to be much more effective than general-purpose languages. Obviously, the more specialized a language is, the smaller its audience is. One way to cope with the communication problem is to adapt the level of specialization according to the interlocutors. In practice, various specialized jargons are organized around a common language. Language customization has been recognized as an important aspect of UML. Since its version 2.0, UML is no longer seen as a "unified" language, but rather as a language family. Various dialects can be defined to handle special requirements leading to UML profiles for databases, for C++ development, for real-time system modeling, etc.

Summing up, many different factors explain the "babelization" in the software community. At the time of writing this paper, there are more than 8000 programming languages referenced in HOPL [15]. If the DSL trend is successful, then this number is expected to grow dramatically. One single language does not fit all needs. Even if this were the case, there would still be a multiplicity of languages and variants due to political and social pressure. Thus, we must live with this diversity. MDE presented next is intended to support and address this diversity.

3. MODEL DRIVEN ENGINEERING

In this section, we introduce the key concepts and principles necessary to understand the essence of MDE: technological spaces, models, metamodels, transformations and mappings.

3.1. Technological spaces

As discussed above, 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" [16]. 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, model transformations, and mappings, offer the appropriate means.

3.2. 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" [17]. Thus, a model, built to address one specific aspect of a problem, is by definition a simplification of the actual thing. 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.

The model is a role of representation that a system plays for another one. Models form oriented graphs (graphs) whose edges denote the relation "is represented by" (Fig. 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 [18] are contemplative models of human experience in a specified setting. As discussed in Section 2, they support human thinking as well as human-to-human communication. On the other hand, task models exploited in TERESA [19] 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 [19] task model is compliant with CTT [10]. A language is the set of all well-formed expressions that comply with a grammar (along with its 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 (relation "is part of" on Fig. 2), it is called a meta-model. CTT [10] is a meta-model for expressing specific task models.

3.3. Metamodels

A metamodel 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 tree: a model complies to a single metamodel, whereas a metamodel may have multiple compliant models. In the same way, a meta-metamodel is a model of a set of metamodels that are compliant with it. It does not represent metamodels, but sets the rules for producing distinct metamodels.

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-metamodel. This top level is called M3, giving rise to metamodels, 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 and 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 are the MDE answers to these issues.

3.4. Transformations

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 [20]. Source and target models are related by the relation "is transformed into". Note that a

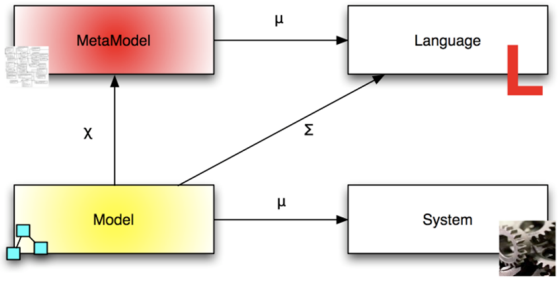


Figure 2: *Basic concepts and relations in MDE.*

set of transformation rules is a model (a transformation model) that complies with a transformation metamodel.

Transformations can be characterized within a four - dimension space: A 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. A transformation is vertical when the source and target models reside at different levels of abstraction. 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. 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 metamodel). 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.

Relation 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.

3.5. Mappings

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 running UI. The correspondence between the source task (and concepts) and their target widgets is maintained as mappings. In summary, we must live with the "babelization" of the software landscape. Fortunately MDE is an approach to cope with this diversity. As discussed next, it can be usefully exploited for the many faces of plastic multimodal UIs.

4. THE MANY FACES OF PLASTIC MULTIMODAL UIS

Designing and developing UIs involve multiple stakeholders such as computer scientists and human factor specialists, whose background, priorities and role in the development process are quite different. As discussed in Section 2, they belong to distinct communities, and thus use different languages and approaches. Although practices differ between HCI and software communities, we have observed a number of recurring steps that now serve as references for designing and developing plastic multimodal UIs. This reference framework is presented in sub-section 4.2 illustrated with a simple example: a Home Heating Control System (HHCS).

4.1. Case Study: HHCS

HHCS enables end 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. 3, each UI solution depends on the device screen size, as well as on the set of usability properties that HCI designers have elicited as key.

From a functional perspective, the four UI's of Fig. 3 are equivalent: they support the same set of tasks, providing access to the same set of rooms (the living room, the cellar and the kitchen) where the room temperature may be set between 15C and 18C.

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 [21], prompting (a factor for guidance) and protection (a factor for error control) are not equally supported. In Fig. 3-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 supported, and the risk for input error is high. In Fig. 3-b), the lack of prompting is repaired but the user is still not prevented from entering wrong values. Solution in Fig. 3-c satisfies both criteria.

Figure 4: *A hand-made mock-up similar to Fig. 3-d where a clickable map of the house improves Compatibility.*

Other criteria such as task compatibility, homogeneity-consistency, and workload could be called upon as well. Fig. 4 shows an alternative to Fig. 3-d where a clickable map improves compatibility. In terms of workload, the pull-down menus proposed in Fig. 3-c are more appropriate than the input text fields used in Fig 3-a and Fig. 3-b (since less actions are required).

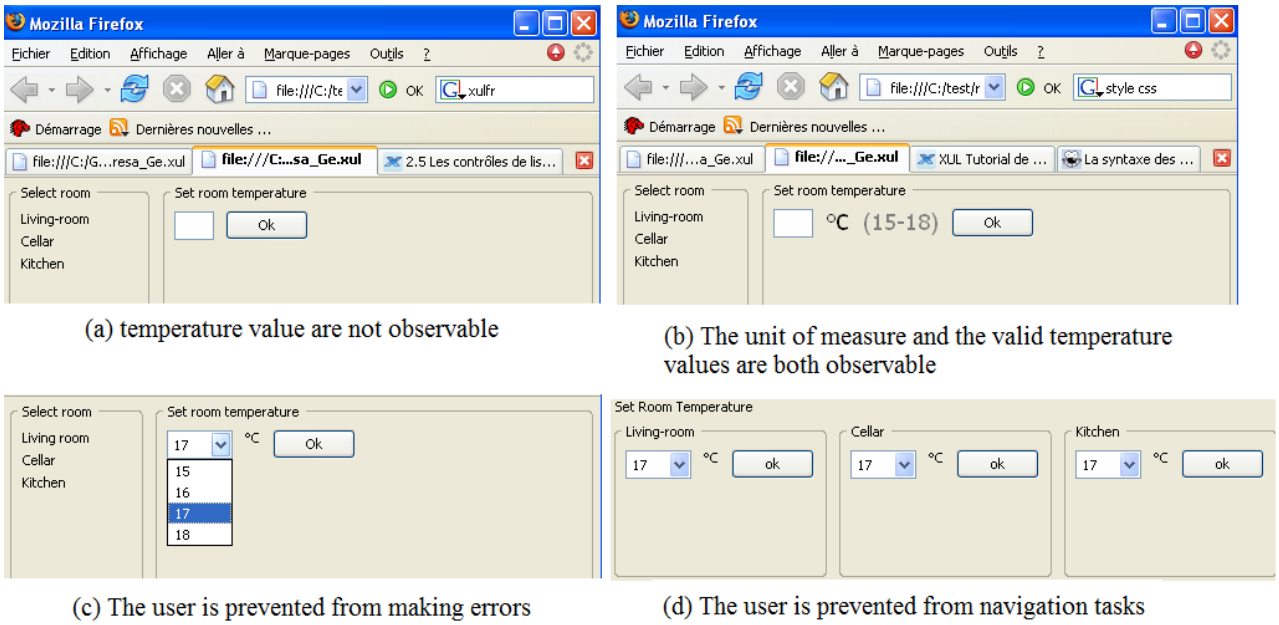


Figure 3: *Four functionally-equivalent UI's that differ from the set of usability criteria used to produce them.*

The UIs of Fig. 3-a-b-c reflect the task decomposition: one dialogue space has been defined per task. As such, they are task-driven whereas the UIs shown in Fig. 3-d and Fig. 4 are concept-driven (user's tasks are now viewed as operations on concepts). Here, the Grouping/Distinction among items has been considered as a key criterion. As a result, there is one dialogue space per room in charge of supporting all the tasks that are applicable to the concept.

Task driven UIs can also serve as a basis for re-distributing UI portions across the interaction resources that are currently available in the context of use. Fig. 5 shows an example where the task Select a room has migrated to a PDA whereas the task Set room Temperature has been kept on the PC. The redistribution has triggered a remolding where hyperlinks have been replaced with radio-buttons.

In this example, UI redistribution has been controlled by the end-user through a meta-UI [22]: the end-user has selected the task Select room from a representation of the task model (left side of Fig. 6) followed by the specification of the platform onto which the selected task should be executed (here, the PDA). Denoting the target platform is performed by selecting the appropriate element of the platforms list (bottom right of Fig. 6). This list serves as a representation of the platform model.

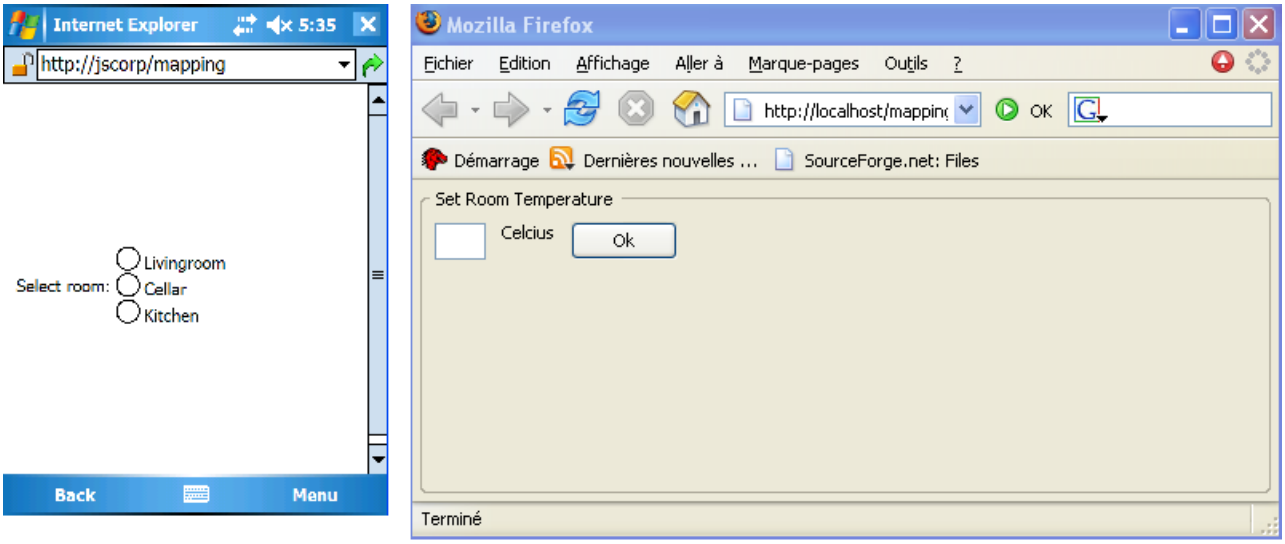
The Meta-UI also supports the modification of the tasks type. For instance, the end-user can decide that the task Select a room is of type "Input" rather than "Choice of one option among N". This would trigger a remolding of the UI that would replace the combo boxes of Fig. 3-c with the input field of Fig. 3-a.

The design of a UI whether it is (meta- or not) follows the reference framework described next.

4.2. Reference Framework for Plastic Multimodal User Interfaces

The framework shown in Fig. 7 is a canonical decomposition that structures the development process for plastic multimodal UIs [23]. It includes four models related by transformations that each corresponds to a set of concerns (and stakeholders) so as to distribute the development life cycle over several levels of abstractions, thus coming up with multiple models [24]:

- The Tasks & Concepts (T & C) model describes the tasks to be carried out by the end-users as well as the domain-dependent concepts manipulated in the execution of these tasks. This model is typically produced by human factors specialists. Fig. 6 shows a representation of the task model for HHCS.
- The Abstract UI (AUI) model defines dialog spaces by grouping subtasks according to criteria such as task model structural patterns, cognitive workload, semantics relationships between domain-dependent concepts. Dialog spaces are related by a navigation scheme that reflects tasks relationships. They are populated with Abstract Interaction Objects (AIOs) [25]. AIOs represent domain-dependent concepts in a modality - independent way . An AUI is an abstraction of a Concrete UI (CUI) with respect to modality. However, dialog spaces may be modality-dependent as they can embed mechanisms for solving references and ambiguities. This is typically the case for natural speech-based UIs. For example, two dialogue spaces have been defined for HHCS in Fig. 3-a-b-and c: one for selecting the appropriate room, and a second one for setting the temperature of the selected room. The AIO Select I among N supports the navigation task between the dialogue spaces. In Fig. 3-d, dialogue spaces have been defined in a concept-oriented way resulting in the absence of explicit navigation tasks.
- The Concrete UI (CUI) model makes an AUI concrete for a given context of use where AIOs are replaced with Concrete Interaction Objects (CIOs) [25] related by layout and navigation constraints. A CIO is modality-dependent. In conventional GUI paradigm, it is called a widget. For example, the AIO Select I among N has been replaced with a combo box in Fig. 3-a-b-and c. Although a CUI shows the final look & feel, it is still a mock-up that runs within a particular environment. A CUI is a reification of an AUI. It is also an abstraction of the FUI with respect to the execution platform. In general, CUIs are produced



(a) "Select a room" has migrated to the PDA

(b) "Select a room" has disappeared from the PC

Figure 5: *A distributed version of HHCS: selecting a room is performed on the PDA while the temperature is specified on the PC.*

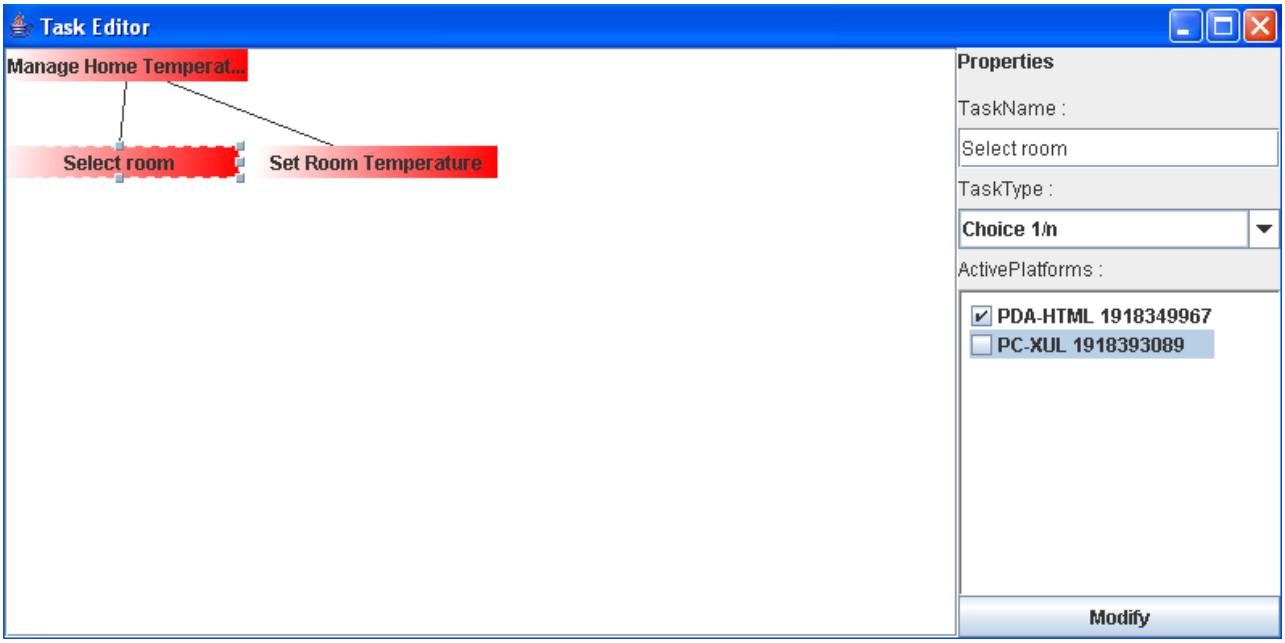


Figure 6: *How to ask for dynamic UI migration by assigning tasks from the task model to platforms of the platform model.*

by designers with the help of human factor specialists.

- The Final UI (FUI) is the operational UI, i.e. a UI that runs on a particular computing platform. It can be interpreted (e.g., by a Web browser) or executed (as the result of code compilation). FUIs are produced by software developers.

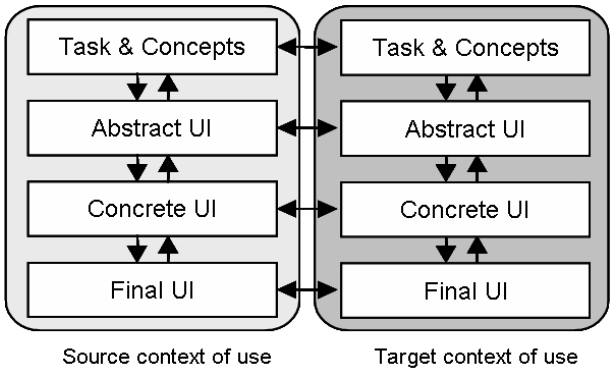


Figure 7: *The CAMELEON Reference Framework [23].*

These models are related by transformations and mappings. The framework shows two types of vertical transformations (abstraction and reification), as well as horizontal transformations (translations). For example, for the GUI modality, rendering is a transformation where tasks and concepts are transformed into dialogue spaces, which, in turn, are transformed into windows populated with graphical widgets.

Transformations can be combined and applied to any of these models to support UI adaptation using any entry point in the reference framework. We call this multi-path UI development. As an example, using VAQUITA [26] and WebRevEnge [27], one can reverse engineer HTML source files into more abstract descriptions (respectively CUI and AUI, task), and from there, either retarget and generate the UI or apply a combination of retargeting and/or forward engineering tools. This means that designers and 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. In the next section, we illustrate the exploitation of this flexibility with a set of tools developed according to this framework.

5. TOOLS AND POWERFUL GATEWAYS FOR PLASTIC MULTIMODAL USER INTERFACES

This section presents a short survey of tools that can be used for the development and/or execution of multimodal plastic UIs. As shown in 5.1, each tool is intended to solve a small set of issues. As a result, tools must be interoperable in order to cover the development and execution of plastic multimodal UIs appropriately. The concept of gateway between tools is illustrated in 5.2.

5.1. Representative Tools for Multimodal Plastic UIs

MONA (Mobile multimOdal Next generation Applications) is an environment for producing Multimodal UIs. Although complete, it addresses the specific case of Web-based applications. It involves a presentation server for a wide range of mobile devices using wireless LAN and mobile phone networks that transforms a Multimodal Web UI (MWUI) specification into a graphical or multimodal UI than can adapt to diverse devices dynamically: WAP-phones, Symbian-based smart phones or PocketPCs and PDAs. The application design process is based on use cases that allow, for each device, the refinement and validation of the design of multimodal UI prototypes. These prototypes are then submitted to a heuristics-based evaluation performed by evaluators with design experience.

ICARE [28] is a component-based approach to the design and development of multimodal UIs, composed of elementary components. An elementary component supports a pure modality (e.g., speech only, graphics only). Components are assembled according to the CARE properties [29]: complementarity, redundancy, and equivalence using a graphics editor. An assembly built with ICARE is then automatically transformed into executable code. However, at run-time, this code is unable to adapt dynamically to the context of use. In addition, multimodality is limited to inputs whereas MOST is limited to outputs.

MOST (Multimodal Output Specification Platform) [30] is a platform that allows the design of output multimodal systems (i.e. graphical, vocal and tactile modalities) based on a three-step process: analysis, specification and simulation. In the analysis phase, the output interaction components (i.e., mode, modality and medium) are identified. The specification step formalizes the results of the previous step based on a series of attributes and criteria assigned to each specific output interaction component. Depending on the current state of the interaction context, a behavioral model allows the identification of the most suitable output form that can be used in order to present each interaction component. The behavioral model is a set of selection rules that produces the appropriate multimodal presentation.

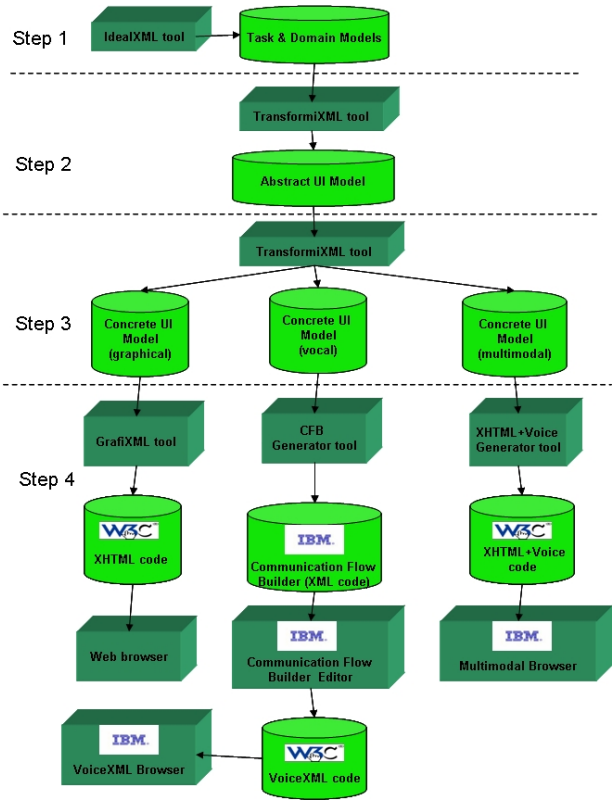


Figure 8: *A global picture of the method, language and tools used in MultimodaliXML. It shows a production chain of interoperable tools based on a common underlying language: UsiXML.*

TERESA [19] automatically generates X+V MWUIs using

the framework presented in Section 3: the initial task model for the envisioned system is transformed into a system task model that is specific to the target multimodal platform. The system task model is in turn transformed into an abstract UI, a concrete UI, then into the code of the final UI. However, the transformation process uses parameters that are not related into a coherent and explicit set of design options. In addition, TERESA transformations are hard coded and embedded into the code, whereas they are made explicit, thus browsable and modifiable in MultimodaliXML [31] as well as in MARA [9].

As opposed to the tools presented so far, MultimodaliXML [31] brings together the following issues into an integrated design space: the use of models to produce multimodal interfaces (e.g., [32]), languages for models specification (e.g., DISL [33], D3ML [34], RIML [35], UIML [36], MXML http://www.macromedia.com, XISL [37]), and explicit design options for multimodal dialog (e.g., CARE properties [29], task-based design of multimodal applications [19]).

MultimodaliXML design options for MWUIs [38] are structured according to three types of pure modalities (graphical, vocal, tactile) and the combination of them. These design options provide designers with explicit guidance for their future UI, and allow them to explore design alternatives. They have been implemented as graph transformations performed on a UI model. Thanks to a transformation engine, designers play with different values for each design option to preview the results of the transformation, and to obtain the corresponding code on demand. The entry point of the process is the "Tasks & Concepts" level. Transformations are reifications that take into consideration tuned ergonomic criteria for MWUIs. These ergonomic criteria were primarily defined for the evaluation of UIs [21]. Although MultimodaliXML suggests certain coverage of the reification process, the underlying hypothesis is that all of the models are expressed using one single User Interface Description Language (UIDL): UsiXML, which stands for User Interface eXtensible Marlup Language (http://www.usixml.org).

UsiXML covers the definition of UIs at any level of abstraction as well as the definition of transformation rules for addressing specific constraints. It is gradually enriched for the design of MWUIs. In practice, UsiXML is supported by a suite of tools including editors, generators, interpreters, and models managers. These tools interoperate as they are based on the same language: UsiXML. Fig. 9 shows the overall picture of the UsiXML tools.

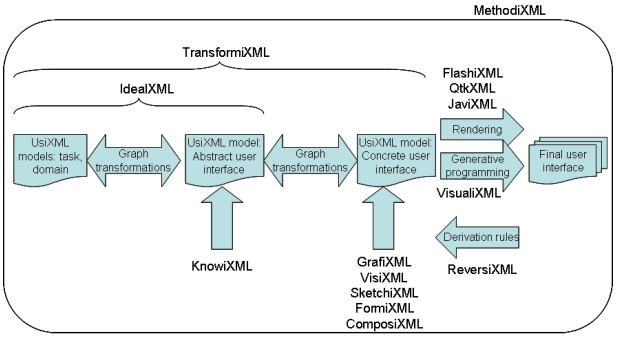


Figure 9: *An overview of the UsiXML suite of of tools.*

The UsiXML-based tools are intended to cover the design phase of UIs, not the running phase. For instance, SketchiXML [39] (see Fig. 10) is a multi-agent interactive editor that enables designers and end-users to sketch parts or all of the UIs with different levels of details and support for different contexts of use. SketchiXML is intended to convey informal specifications of the UI presentation and dialogue. The behaviour of the application can be tuned with a set of parameters. As an example, the designer is free to define whether a recognized shape should appear differently than an unrecognized shape, or to define the level of fidelity required. The sketch of a UI is then analyzed to produce UI specifications independently of any context, including user and platform (UsiXML or UIML). These specifications are exploited to progressively produce one or several UIs, for one or many users, platforms, and environments. In any case, SketchiXML only provides the core properties of these different components since this kind of low fidelity design tool is not supposed to capture detailed information. The next step of the design process consists in importing the specifications generated with SketchiXML into a high-fidelity editor (GrafiXML for UsiXML and LiquidUI for UIML). With these tools, the designer is then able to specify the attributes that could not be specified during the sketching phase.

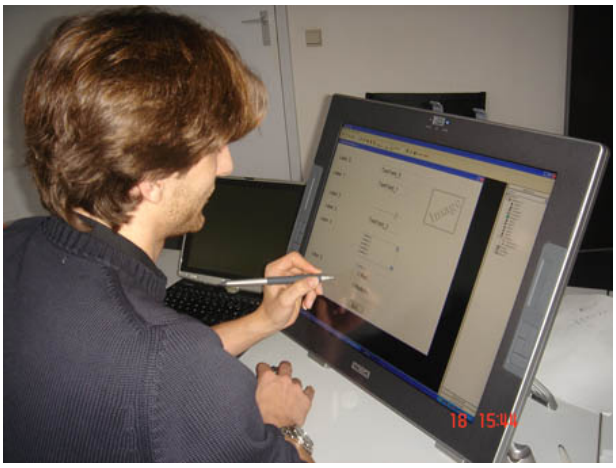


Figure 10: *SketchiXML in action. The user (designer) can draw (sketch) low-fidelity shapes (as in Peridot [40]) that can be recognized as widgets or left as is, depending on the designers's intent.*

Reusability calls for tools that are able to support UI fusion and fission. ComposiXML [41] has been designed to that end. ComposiXML is a plug-in developed for the GrafiXML editor. Any individual or composite component of a graphical UI is submitted to a series of operations for composing a new UI from existing components and for decomposing an existing one into smaller pieces that can be used in turn for another UI. The composition and decomposition operations are defined by the way of the tree algebra. Two kinds of operators are supported: unary operators such as selection and projection, and binary operators such as union, intersection, and fusion (Fig. 11). Composition and decomposition can be performed both at design time and run time.

Most tools presented so far imply a dichotomy between the design phase and the run time phase. Given that in ubiquitous computing, not all situations can be envisioned at design time, the design phase and the run-time phase must be more tightly coupled. The MARA infrastructure [9] is intended to blur this distinction. In MARA, all of the models (that cover the four levels of the reference framework presented in Section 4) are alive at run time, and linked together by the way of mappings. Mappings express ergonomic criteria that drive models transformations. Models (including mappings) are compliant with explicit meta-models. As exemplified next, it is possible to transform MARA meta-models into the UsiXML technological space so as to take advantage of the UsiXML suite of tools.

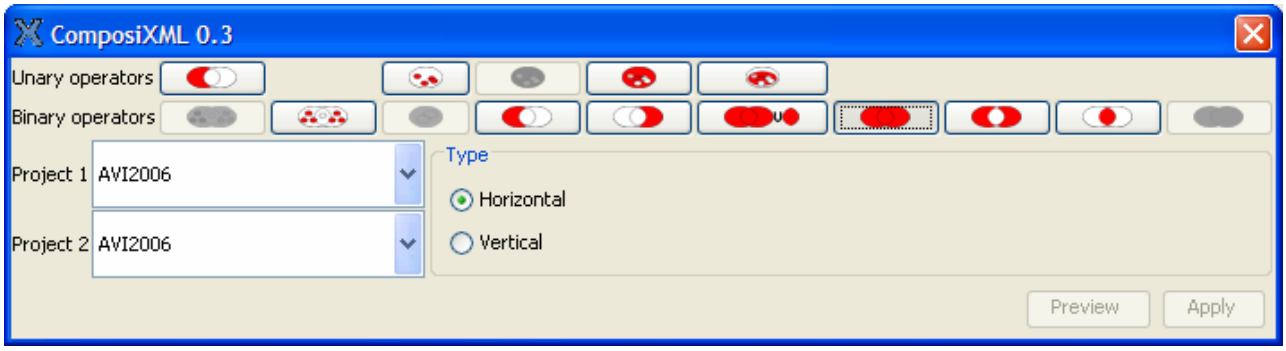


Figure 11: *ComposiXML provides the designer with both unary and binary operators for composing and decomposing UIs.*

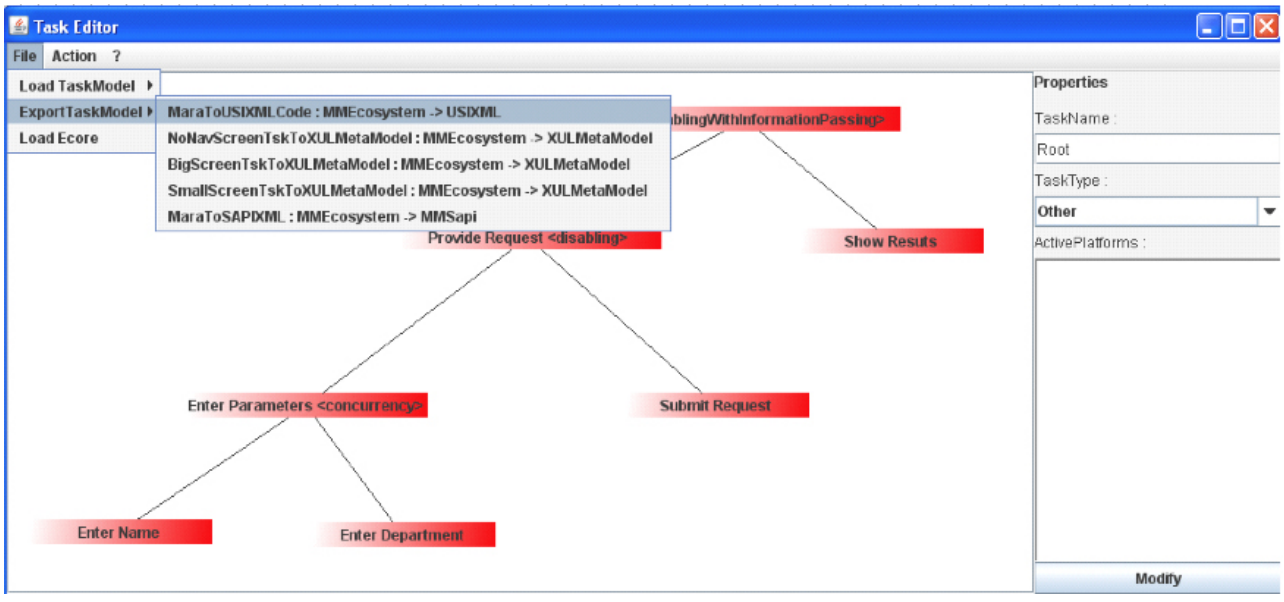


Figure 12: *The MARA Model exporter. Here, the developer has selected to transform the current MARA task model to UsiXML.*

5.2. The Power of Gateways

To demonstrate the power of gateways, we show that it is possible to switch between the MARA infrastructure and the UsiXML-based tools. From a technological perspective, these gateways are implemented as transformations expressed in ATL [42]. These transformations first convert the EMF MARA metamodels into EMF UsiXML metamodels that in turn, are transformed into XML (Fig. 13). These transformations apply at the "Tasks & Concepts" level as well as at the AUI level. When applied at the Task level (Fig. 13), they make it possible to transform a MARA task model into a UsiXML task model and from there, to invoke IdealXML to generate a VoiceXML UI. This facility may prevent the MARA developer from defining specific ATL transformations for targeting VoiceXML. Similarly, task-to-task transformations have been developed to switch from MARA to CTT [10], and from there, to call upon the Teresa environment [19].

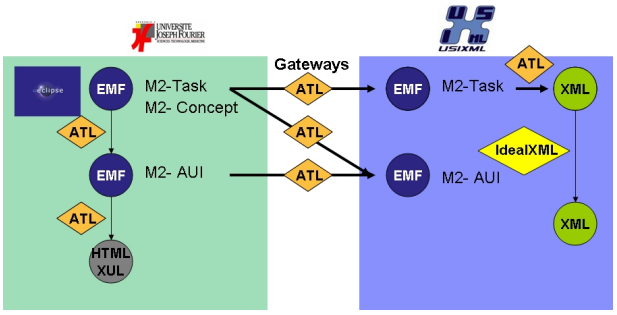


Figure 13: *Calling the appropriate MARA exporter (gateway) to transform a MARA task model into a UsiXML task model, to exploit IdealXML. Alternatively, a MARA task model or a MARA AUI can be transformed into a UsiXML AUI.*

From our early experience with transformations, gateways open the way to a diversity of improvements. First, gateways make it possible for developers and designers to use their favorite tools in conformity with the low-threshold principle. Alternatively, developers and designers may play with the functional coverage of the tools for high ceiling output. This may range from preferring CTTE to the simplistic MARA editor for fine tuning task models, or using SketchiXML for drawing low-fidelity UI's, to switching to advanced tools. For example, switching to a tool that supports design by reuse (as in ComposiXML), switching to a tool that covers modalities not supported by the preferred tool (e.g., from MARA to IdealXML), or to a tool that supports dynamic UI remolding and UI redistribution (as in MARA).

6. CONCLUSION

One important lesson drawn from Linguistics is that "babelization" is unavoidable. Two parallel processes are en route - divergence and convergence - making it necessary for communities, and their languages, to live together. Many Domain Specific Languages are emerging while a significant amount of efforts is put on standardization. This observation holds both for mainstream software engineering and HCI. Software engineering has developed MDA/MDE as a way to cope with the diversity of languages and tools. As for HCI, a model-based approach to UI generation has been promoted since the mid-eighties, thus long before software engineering discovered MDA/MDE. On the other hand, model-based tools have not found wide acceptance. We think that there are three reasons for this.

First, in conventional model-based approaches to UI generation, transformation rules are diluted within the tools. Consequently, "the connection between specification and final result can be quite difficult to control and to understand" [12]. In our approach, transformations are promoted as models. As any model, they can be modified both at design-time and runtime. The same holds for mappings. In particular, mappings are decorated with properties to convey usability requirements.

Second, the conventional model-driven approach did not foresee the benefits from blurring the distinction between the design phase and the run time phase. In particular, the possibility for design tools to become run-time services that can reason at the appropriate level of abstraction or that can be used by end-users to control the rendering and distribution of the final UI, provide new forms of flexibility that could not be imagined before [43]. In addition, the dynamic plug-in of hand-written code makes it possible for designers and developers to recruit pieces of UI that cannot be derived automatically (such as the UI shown in Fig. 4) or that go beyond traditional UIs such as post-WIMP and tangible UIs. By doing so, we alleviate the risk that generated UIs be not as good as those created with conventional programming techniques.

Third, the existence of gateways between languages and tools is a way to reconcile the requirements for tools with low-threshold and high-ceiling: ideally, designers, developers, and end-users can use the languages and tools they know about (low-threshold), then switch to high-ceiling tools by the way of transformations only when these show clear benefits.

To summarize, we claim that MDE coupled with blurring the distinction between the design and run-time phases offers a promising approach to the design, development and execution of plastic multimodal UIs. Metamodels and transformations are now at the heart of the solution space. As knowledge improves with experience, the next step is to capitalize metamodels and transformations into databases (cf. the ZOOOMM project - http://www.zooomm.org) as well as designing UIs for metamodelling and transformations editing, leading to the notion of Mega-UI [44].

7. ACKNOWLEDGEMENTS

This work has been supported by the SIMILAR network of excellence (http://www.similar.cc), the European research task force creating human-machine interfaces similar to human-human communication of the European Sixth Framework Programme (FP6-2002-IST1-507609) as well as by the E-MODE ITEA-1 project.

8. REFERENCES

[1] M. Weiser, "The computer for the 21st century", *Scientific American*, vol. 265, pp. 94–104, September 1991. 1

[2] N. Barralon and J. Coutaz, "Coupling Interaction Resources in Ambient Spaces: There is More than Meets the Eye", in *Proc. of Engineering Interactive Systems 2007 (IFIP WG2.7/13.4 10th Conference on Engineering Human Computer Interaction jointly organized with IFIP WG 13.2 1st Conference on Human Centred Software Engineering and DSVIS - 14th Conference on Design Specification and Verification of Interactive Systems) EIS'2007* (M. B. Harning and J. Gulliksen, eds.), Springer-Verlag, Berlin, March 2007. 1

[3] R. Jacob, A. Girouard, L. Hirshfield, M. Horn, O. Shaer, E. T. Soloway, and J. Zigelbaum, "CHI2006: What is the

Next Generation of Human-Computer Interaction?", *Business leadership and the UX manager*, vol. 14, pp. 53–58, 2007. 1

[4] J. Coutaz and G. Calvary, *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, ch. HCI and Software Engineering: Designing for User Interface Plasticity. Lawrence Erlbaum Associated Publishers, Mahwah, 2007. 1

[5] D. Thevenin and J. Coutaz, "Plasticity of User Interfaces: Framework and Research Agendas", in *Proceedings of Interact'99* (A. Sasse and C. Johnson, eds.), pp. 110–117, IOS Press, Amsterdam, 1999. 1

[6] B. Ullmer and H. Ishii, *Human-Computer Interaction in the New Millenium*, ch. Emerging Frameworks for Tangible User Interfaces. Addison-Wesley, Reading, 2001. 1

[7] K. P. Fishkin, T. P. Moran, and B. L. Harrison, "Embodied User Interfaces: Toward invisible User Interfaces", in *Proceedings of the IFIP TC2/TC13 WG2.7/WG13.4 Seventh Working Conference on Engineering for Human-Computer Interaction EHCI'98*, pp. 1–18, Kluwer Academics Publishers, Deventer, 1998. 1

[8] C. Coutrix and L. Nigay, "Mixed Reality: A Model of Mixed Interaction", in *Proceedings of International Conference on Advanced Visual Interfaces AVI'2006*, pp. 43–50, ACM Press, New York, 2006. 1

[9] J. S. Sottet, G. Calvary, J. Coutaz, and J. Favre, "A Model-Driven Engineering Approach for the Usability of Plastic User Interfaces", in *Proc. of Engineering Interactive Systems 2007 (IFIP WG2.7/13.4 10th Conference on Engineering Human Computer Interaction jointly organized with IFIP WG 13.2 1st Conference on Human Centred Software Engineering and DSVIS - 14th Conference on Design Specification and Verification of Interactive Systems) EIS'2007* (M. B. Harning and J. Gulliksen, eds.), Springer-Verlag, Berlin, 2007. 2, 8

[10] F. Paternó, C. Mancini, and S. Meniconi, "ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Modelss", in *Proceedings of Interact'97*, pp. 362–369, Chapman & Hall, London, 1997. 2, 3, 10

[11] F. Gamboa-Rodriguez and D. L. Scapin, "Editing MAD* task descriptions for specifying user interfaces, at both semantic and presentation levels", in *Proceedings of 4th International Eurographics Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'1997*, pp. 193–208, Springer-Verlag, Berlin, 1997. 2

[12] B. Myers, S. E. Hudson, and R. Pausch, "Past, Present, Future of User Interface Software Tools", *ACM Transactions on Computer-Human Interaction*, vol. 7, no. 1, pp. 3–28, 2000. 2, 10

[13] D. Crystal, *The Cambridge Encyclopedia of Language*. Cambridge University Press, Cambridge, second edition ed., 2005. 2, 3

[14] R. Bijeljac and R. Breton, *Du language aux langues*. Gallimard, Paris, 1997. 2

[15] D. Pigott, "HOPL: an Interactive Roster of Programming Languages", 2007. http://hopl.murdoch.edu.au/. 3

[16] I. Kurtev, J. Bézivin, and M. Aksit, "Technological Spaces: An Initial Appraisal", in *Proceedings of the Confederated International CoopIS, DOA, and ODBASE 2002*, pp. 231–244, Springer-Verlag, Berlin, 2007. 3

[17] J. Bézivin, "In Search of a Basic Principle for Model Driven Engineering", *The European Journal for the Informatics Professional*, vol. 2, pp. 21–24, 2004. 3

[18] M. B. Rosson and J. M. Carroll, *Usability Engineering: Scenario-Based Development of Human-Computer Interaction*. Morgan Kaufman, San Francisco, 2002. 3

[19] S. Berti, F. Correani, G. Mori, F. Paternó, and C. Santoro, "TERESA: A Transformation-Based Environment for Designing Multi-Device Interactive Applications", in *Proceedings of ACM International Conference on Human Factors in Computing Systems CHI'04*, pp. 793–794, ACM, ACM Press, New York, 2004. 3, 7, 8, 10

[20] T. Mens, K. Czarnecki, and P. V. Gorp, "A Taxonomy of Model Transformations", in *Language Engineering for Model-Driven Software Development*, March 2004. 3

[21] J. M. C. Bastien and D. Scapin, "Ergonomic Criteria for the Evaluation of Human-Computer", Tech. Rep. 156, INRIA, 1993. 4, 8

[22] J. Coutaz, "Meta-User Interface for Ambient Spaces", in *Proceedings of the 5th International Workshop on Task Models and Diagrams for UI design TAMODIA'2006* (K. Coninx, K. Luyten, and K. Schneider, eds.), pp. 1–15, Springer-Verlag, Berlin, 2007. 5

[23] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt, "A Unifying Reference Framework for Multi-Target User Interfaces", *Interacting With Computers*, vol. 15, no. 3, pp. 289–308, 2003. 5, 7

[24] J. Vanderdonckt, E. Furtado, V. Furtado, Q. Limbourg, W. Silva, D. Rodrigues, and L. Taddeo, *Multi-model and Multi-level Development of User Interfaces*, ch. Multiple User Interfaces - Cross-Platform Applications and Context-Aware Interfaces. John Wiley & Sons, New York, November 2003. 5

[25] J. Vanderdonckt and F. Bodart, "Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection", in *Proceedings of the ACM Conference on Human Factors in Computing Systems INTERCHI'93*, ACM Press, New York, 1993. 5

[26] L. Bouillon and J. Vanderdonckt, "Retargeting Web Pages to other Computing Platforms", in *Proceedings of IEEE 9th Working Conference on Reverse Engineering WCRE'2002*, pp. 339–348, IEEE Computer Society Press, Los Alamitos, 2002. 7

[27] L. Paganelli and F. Paternó, "Automatic Reconstruction of the Underlying Interaction Design of Web Applications", in *Proceedings of the 14th ACM International Conference on Software Engineering and Knowledge Engineering SEKE'02*, pp. 439–445, ACM Press, New York, 2002. 7

[28] J. Bouchet and L. Nigay, "ICARE: a Component-based approach for the design and development of multimodal interfaces", in *Proceedings of the ACM International Conference on Human Factors in Computing Systems CHI'2004*, pp. 1325–1328, ACM, ACM Press, New York, 2004. 7

[29] J. Coutaz, L. Nigay, D. Salber, A. Blandford, J. May, and R. Young, "Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE properties", in *Proceedings of 5th IFIP TC 13 International Conference on Human-Computer Interaction INTERACT'95* (K. Nordbyn, P. H. Helmersen, D. J. Gilmore, and S. A. Arnesen, eds.), pp. 115–120, Chapman & Hall, London, 1995. 7, 8

[30] C. Rousseau, Y. Bellik, and F. Vernier, "Multimodal Output Specification/simulation Platform", in *Proceedings of 7th International Conference on Multimodal Interfaces ICMI'2005*, pp. 84–91, ACM Press, New York, 2005. 7

[31] A. Stanciulescu, Q. Limbourg, J. Vanderdonckt, B. Michotte, and F. Montero, "A Transformational Approach for Multimodal Web User Interfaces based on UsiXML", in *Proceedings of 7th International Conference on Multimodal Interfaces ICMI'2005*, pp. 259–266, ACM Press, New York, 2005. 8

[32] M. Beaudouin-Lafon, "Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces", in *Proceedings of the ACM Conference on Human Factors in Computing Systems CHI'2000*, pp. 446–453, ACM, ACM Press, New York, April 2000. 8

[33] R. Schaefer, S. Bleul, and W. Mueller, "Dialog modelling for multiple devices and multiple interaction modalities", in *Proceedings of the 5th International Workshop on Task Models and Diagrams for UI design TAMODIA'2006* (K. Coninx, K. Luyten, and K. Schneider, eds.), vol. 4385, Springer-Verlag, Berlin, 2007. 8

[34] N. Chevassus, "A Framework for authoring and exploiting multimodal documentation", in *W3C Seminar*, (Toulouse, France), June 21 2005. 8

[35] T. Ziegert, M. Lauff, and L. Heuser, "Device Independent Web Applications- The Author Once - Display Everywhere Approach", in *Proceedings of 4th International Conference on Web Engineering ICWE'04* (N. Koch, P. Fraternali, and M. Wirsing, eds.), vol. 3140, pp. 244–255, Springer-Verlag, Berlin, 2004. 8

[36] M. F. Ali, M. A. Pérez-Quiñones, and M. Abrams, *Multiple User Interfaces - Cross-Platform Applications and Context-Aware Interfaces*, ch. Building multi-platform user interfaces with UIML. John Wiley & Sons, New York, 2004. 8

[37] K. Katsurada, Y. Nakamura, H. Yamada, and T. Nitta, "XISL: A Language for Describing Multimodal Interaction Scenarios", in *Proceedings of 5th International Conference on Multimodal Interfaces ICMI'2003*, pp. 281–284, ACM Press, New York, 2003. 8

[38] A. Stanciulescu and J. Vanderdonckt, "Design Options for Multimodal Web Applications", in *Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'06*, pp. 41–56, Springer-Verlag, Berlin, 2007. 8

[39] A. Coyette, S. Kieffer, and J. Vanderdonckt, "Multi-Fidelity Prototyping of User Interfaces", in *Proceedings of 11th IFIP TC 13 International Conference on Human-Computer Interaction INTERACT'2007*, Springer-Verlag, Berlin, 2007. 8

[40] B. A. Myers, "Creating user interfaces using programming-by-example, visual programming, and constraints", *ACM Transactions on Programming Languages and Systems*, vol. 12, no. 2, pp. 143–177, 1990. 8

[41] S. Lepreux and J. Vanderdonckt, "Towards Supporting User Interface Design by Composition Rules", in *Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'06*, Industrial track, Irvine, 2002. 8

[42] F. Jouault and I. Kurtev, "Transforming Models with ATL", in *Satellite Events at the MoDELS 2005 Conference: MoDELS 2005 International Workshops OCLWS, MoDeVA, MARTES, AOM, MTiP, WiSME, MODAUI, NfC, MDD, WUsCAM* (J. M. Bruel, ed.), vol. 3844, pp. 128–138, Springer-Verlag, Berlin, 2006. 10

[43] L. Balme, A. Demeure, N. Barralon, J. Coutaz, and G. Calvary, "CAMELEON-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces", in *Proc. of Second European Symposium on Ambient Intelligence EUSAI'2004* (P. Markopoulos, B. Eggen, and E. Aarts, eds.), pp. 291–302, Springer-Verlag, Berlin, November 2004. 10

[44] J. Sottet, G. Calvary, J. Favre, and J. Coutaz, *Megamodeling and Metamodel-Driven Engineering for Plastic User Interfaces: Mega-UI*. Springer-Verlag, Berlin, 2007. 10