Integrating AMI, MDE for Muti-device Muli-Modal User Interface to Support People with Disabilities

Eman M. Saleh
Cairo University, Egypt
Eman_maghary@yahoo.com

Aly Fahmy
Prof. of Computer Science
Faculty of Computers and Information, Cairo University, Egypt
Cairo University, Egypt
a.fahmy@fci-cu.edu.eg

Amr Kamel
Cairo University, Egypt
a.Kamel@fci-cu.edu.eg

Abdel Badeeh M. Salem
Prof. of Computer Science
Head of Medical Informatics and eHealth Reserch Unit,
Ain Shams University, Egypt
absalem@asunet.shams.edu.eg

ABSTRACT

Model-driven engineering (MDE), including Model Based approaches of user interfaces describes the User interface and its related aspects (e.g. tasks, domain, context of use) using set of models that can derive a final user interface. Model-Based user interface approaches give the researchers a new designing methodology that eases the creation of user interface and tackles the problems of producing a new design for every new device and every modality (graphical, vocal, gesture, …)

The design of interactive systems for an ambient intelligent environment (AMI) poses many challenges due to the diverse number of devices, and interaction modalities available in the environment together with restrictions imposed by making the interactive system usable by people with disabilities. This work proposes a framework that integrates model based techniques with AMI to build an applicable solution which can allow the designers to design and develop multi-device and multi-modal user interfaces to fit the emergent needs of people with disabilities in an AMI environment through a number of model-transformations.

Key Words: Software Engineering, AMI, Model-Based User Interface Design, ConcurTaskTrees, Task Model, XML-Based Languages, HCI.

1. Introduction

In Model Based User Interface Design (MBUID), different abstract models highlight different aspects of the user interface independent of details of the target devices. A multi level reification steps will be followed to generate concrete models that "fill in" more specific details towards the presentation of the interface on the target platform. This paper considers three important concepts; context dependent UI design, multi-device and multi-modal user interface design. There is still work to be done to visualize the different models and the influence of model manipulations at run time in order to cope with AMI environment.

Examples of model-based systems are Mobi-D[1], Teresa[3][11], Dygimes[2]. High Level User-interface Description–

Languages (HLUIDL) had been linked extensively to Model-Based user interface development because they offer multi-device UI creation, more specific the XML-Based HLUIDL, because most of them have firmly focused on usability and scalability: making one design for many devices is the main goal, they succeed to achieve this goal for form-based interfaces, but it's not the case for graphical multi-modal interfaces.

Examples of these languages are the UIML[8], RIML[9], TeresaXML[11], useML[5], ISML[7], XIML[6], UsiXML[10] and there are many existing languages differ in their degree of abstraction, model coverage, Standardization and the availability for users.

A relatively new tool, by Stanciulesu et.al., TransformiXML [12] addressed the creation of multimodal web user interfaces through set of model-to-model transformations which transform a UsiXML compliant specification into another UsiXML. Transformation rules are expressed in UsiXML compliant UI to produce a new UI. The work is restricted to web UI and UsiXML models.

To enable people with disabilities to communicate with applications the same way they naturally do in their daily behavior the UI should be tailored to different user capabilities, taking into consideration the diversity of devices that may be used varying from desktop computers, mobile devices to wearable devices with small screen size and limited interaction devices.

A *multimodal* user interface gives the end users the ability to choose the interaction modality that is the most suitable for their capabilities. A modality can be expresses as a couple (device, interaction device) some of the modalities can be (keyboard, command language), (mouse, direct manipulation), (loudspeakers, unrestricted natural language) [12].

There are a wide range of different models that can be used in Model-Based User Interface Development: *Task model* (a model that describes the goals that the user hopes to accomplish, and the actions that must be taken to accomplish them), *Data* or *Domain model* (a model that describes the objects and data that the user will be concerned with), *Application model, dialog model* (a model that describes the mechanics of how the user is to interact with the UI. It specifies the navigational structure of the UI, and the used interaction techniques), *presentation model* (a model that describes the visual appearance of the user interface. It specifies which widgets have been selected, and where they are placed, among other things) and *user model* (that describes properties of the users themselves, such as their level of expertise, or their security clearance model). The data, domain and application model can be situated at the end of the application logic of the system. They define the type of objects and the operations on objects that can be used or needed to be supported by the interactive system. The task and domain model closest to the user and specify the tasks the user executes and the objects manipulated by the UI. The dialog model and presentation model are closest to the final user interface. The most important model to support AMI environments is the *Context model:* a model that can describe the context-of-use for an interactive system. *In AMI* environment, systems are no longer bound to a single place and situation, the designer should be able to define the possibilities to execute the task represented by UI while taking into account constraints posed by the user's environment. E.g. a context model could specify a set of external parameters that can influence the appearance, usage, modality,… of an interactive system. This model is the least explored, but becomes increasingly important in modern interactive systems.

The remainder of this paper is structured as follows: The next section gives an overview of previous work section 3 introduces the user interface description language: UsiXML , Section 4 presents the proposed framework for a design and runtime architecture, finally section 5

concludes the paper and puts forward some of the future work.

2. Previous work

If we take the definition of MBUID as a set of models, *Mastermind* [19]; is one of the first projects to generate a user interface by combining different models; it used the presentation, application and dialog models to automatically generate the user interface [18][20].

Trident (Tools foR an Interactive Development EnvironmeNT) is a model-based system to create an interactive system. [21][22]. It was one of the first design tools that recognized the importance of a clear separation between an abstract representation of the presentation model and a concrete representation thus supporting a multitude of interaction style alternatives for the same functional core. It also integrated task analysis as an important component to create a usable interface. Together with DON; which is an earlier tool supporting the domain model and integrates the presentation model in its design methodology; Trident can be considered to be one of the first "complete" Model-Based User Interface Development Environments that where available.

Tadeus (Task Analysis/Design/End User Systems) is a Model-Based User Interface Development environment that focuses on a user model, a task model, a domain model, a dialog model and later an interaction model was added [23].

Mobi-D is a model-based integrated development environment that combines several declarative models and assists the user interface designers with the creation of these models and with the decisions they will have to make during the design of the user interface [1]. Mobi-D offers a complete design cycle with a set of tools, and supports iterative refinements in the design of the user interface. Mobi-D works task driven.

Teresa (Transformation Environment for interactive Systems representAtions) [3,13] and *Dygimes* (DYnamically Generating Interfaces for Mobile and Embedded Systems) [2], none of the aforementioned tools addressed the design of interactive system deployed for AMI environment, most recent work in this area focus in a subtopic, Luyten et.at [24] presented an approach to take context switching explicitly into account in the task and dialog model by inserting decision tasks in the task model, Georgantas and Issarny show a functional approach towards modeling a situation sensitive user interface in [29]. Just as in the ICrafter [30] introduced a service framework for user interface services. is created. Most of this work reflects the need for some kind of unified framework to design and develop the interactive part of a computing system that is deployed in an ambient intelligent environment. However, there is *no* design support to constrain the dynamic behavior of these systems so the resulting user interface is usable and still supports the envisioned tasks depending on the situation.

3. Choosing a User Interface Description Language

To support the design and development of Multi-device Multi-modal user interfaces we propose to use UsiXML HLUIDL [10], This choice is based on the coverage of most needed models to support AMI and multi-modal UIs; UsiXML structured according to four basic levels of abstractions defined by the Cameleon reference framework, (Fig.1).[13]. UsiXML relies on a transformational approach that progressively moves from the Task and Concept level to the Final User Interface and the steps of the transformational approach define in a comprehensive way their logic and application [10] this transformational methodology allows the introduction of new development sub-steps, thus ensuring the possibility to explore alternatives for each sub-step and to add new ones, also Usixml support modality independence by describing the abstract UI level as independent of any specific modality. [12].

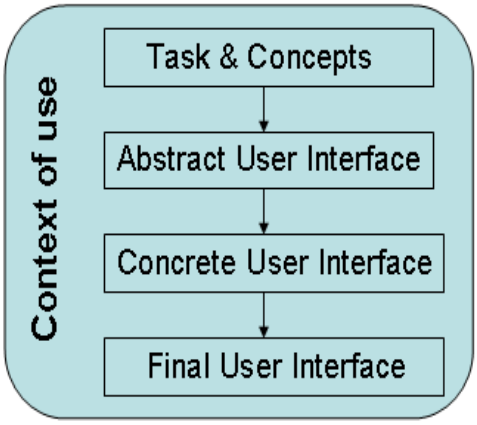

Figure 1: The Cameleon reference framework for multi-target UIs.

Figure 1 outlines the MDE-compliant approach for developing UIs decomposed into four major steps that result from the Cameleon Reference Framework:

1) **Task and domain modeling** (corresponding to the Computing-Independent Model –CIM– in MDE): It models the end user's task, the objects manipulated by application and the context of use (user, platform, and environment). This step is supported by IdealXML [14,15], which models the task model in ConcurTaskTrees (CTT) [13] notation.

2) **Abstract User Interface** (corresponding to the Platform-Independent Model –PIM– in MDE): this level describes the UI independently of any interaction modality and any implementation technology it considered as an abstraction of a CUI with respect to interaction modality (e.g., graphical interaction, vocal interaction, speech synthesis and recognition, video-based interaction, virtual, augmented or mixed reality). It defines how UI Abstract individual components are grouped into set of Abstract Containers [10].

3) **Concrete User Interface** (corresponding to the Platform-Specific Model –PSM– in MDE): this level describes a potential user interface after a particular interaction modality has been selected (e.g., graphical, vocal, multimodal). This step is supported by several tools helping designers and developers to edit, build, or sketch a user interface only for graphical user interfaces for example the UsiXML tool GrafiXML [16]. It concretizes an abstract UI for a given context of use into Concrete Interaction Objects (CIOs) so as to define widgets layout and interface navigation.

4) **Final User Interface** (corresponding to the code level in MDE): this level is reached when the code of a user interface is produced from the previous levels. This code could be either interpreted or compiled. We hereby define a rendering engine as a software component (or set of components) that are able to interpret a UsiXML file expressed at the CUI level and to run it or a code compiler that (semi - automatically generate code from a UsiXML file expressed at the CUI level.

3.1 MODELS in UsiXML:

Before examining closely how MBUID and UsiXML can support AMI for people with disabilities, let us consider the models supported by UsiXML. UsiXML is a collection of models for specifying a UI, some of them being used to support a particular level, some other being used to support a transition from one level to another.

• **Task model**: is a model describing the interactive task as viewed by the end user interacting with the system.

This model uses the ConcurTaskTrees (CTT) by Fabio Paterno, this notation is the most usable and modern specification notation used for task modeling. It provides a graphical syntax, an hierarchical structure and a notation to specify the temporal relation between tasks, an example of CTT task model is shown in figure 2. With this notation, tasks can be classified into four categories: abstract tasks  ,interaction tasks  ,user tasks  and application tasks  . Tasks at the same level can be can be connected by temporal operators like choice ([]), independent concurrency (|||), concurrency with information exchange (|[]|), disabling ([>) , enabling (>>), enabling with information exchange ([]>>), suspend/resume (|>) and order independence (|=|). The precedence of

these operators from highest to lowest are: [] > {|||, |[]|}> {[>,|>} > {>>,[]>>} [28].

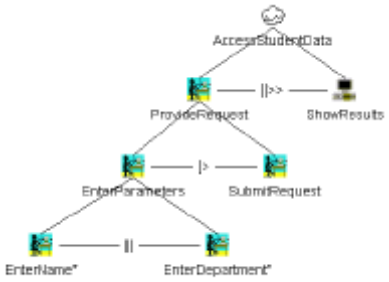

Figure 2.Simple Example of CTT Task Model

• **Domain model**: is a description of the classes of objects manipulated by a user while interacting with a system.
• **Mapping model**: is a model containing a series of related mappings between models or elements of models.
• a**uiModel**: An *Abstract User Interface* (AUI) model is a user interface model that represents a canonical expression of the renderings and manipulation of the domain concepts and functions in a way that is as independent as possible from modalities and computing platform specificities. An AUI is populated by *Abstract Interaction Objects* and Abstract user interface relationships. *Abstract Interaction Objects* (AIO) may be of two types: Abstract Individual Components (AIC) and Abstract Containers (AC). An *Abstract Individual Component* is an abstraction that allows the description of interaction objects in a way that is independent of the modality in which it will be rendered in the physical world. An AIC may be composed of multiple facets. Each facet describes a particular function an AIC may endorse in the physical world.
• **cuiModel**: A *Concrete User Interface* (CUI) model is a UI model allowing a specification of an appearance and behavior of a UI for a given context of uses. A CUI model is composed of Concrete Interaction Objects (CIO) and concrete relationships, which realize an abstraction of widgets sets found in popular graphical and vocal toolkits (e.g., Java AWT/Swing, HTML 4.0, Flash DRK6, Voice-XML, and VoxML). Concrete interaction objects can be further decomposed of concrete graphical objects and concrete vocal objects. A CIO is defined as an entity that users can perceive and/or manipulate (e.g., push button, text field, check box, vocal output, vocal input, vocal menu). The CUI abstracts a Final UI in a definition that is independent of programming toolkit, concrete interaction objects and relationships are further refined into graphical objects and relationships and auditory objects and relationships at the final UI level.
• **Transformation model**: Transformation from one model to another, except from the FUI level.
• **Context model**: The *context model* consists of three submodels: a user model, an environment model and a platform model:
– The *user model* decomposes the user population into user stereotypes, described by attributes such as the experience with the system or with the task, the motivation, etc.
– The *environment model* describes any property of interest of the global environment where the interaction takes place. The properties may be physical (e.g., lighting or bandwidth conditions) or psychological (e.g., level of stress).
– The *platform model* captures relevant attributes related the combination of hardware and software where the user interface is intended to be deployed (e.g PDA with MacOS).
There are five goals that can be considered in MBUID to support AMI[17][2]
Challenge 1: Task-Centered Interfaces
Challenge 2: Multi-Platform Support
Challenge 3: Interface Tailoring
Challenge 4: Multi-Modal Interfaces
Challenge 5: Context-Sensitive Interfaces
The first challenge, have been tackled already by UsiXML support using CTT to describe the task model. The third challenge is implicitly tackled by the way the CUI is defined. In this paper we focus on the other three challenges (2 ,4 and 5), more specifically Context sensitive interfaces because it's the core in an AMI area.

4. The Proposed Design and Generation Process:

We propose a framework to support the creation of context sensitive Multi-device multi-modal user interfaces, our method relies on the separation of concepts, in this way changing any sub-model of the context model, will not change the application code, only the user interface presented to the user will change according to change in context. We propose a reification schema of context information based on transformations, figure 3.

We added enhancements to the current modes to cope with the change of context at run time, so models have to be dynamic instead of static, our focus is strict to the context model, for the context model to be dynamic we introduce two new objects, Abstract Context Objects (ACO) and Concrete Context Objects (CCO), these two types of objects are analogue to AIC and CIC of UsiXML, these objects will support the link of environment changes to navigation and presentation of the user interface.

In summary the proposed design process consists of the following steps:

Step1: Create the dynamic task model based on CTT. The current version of the tool models how the goal of the user is reached by set of tasks but does not support modeling dynamic context changes, since in an AMI environment the execution of the task is highly dependent in the situation of the user and the environment in which the task is executed, and to link the task model with the abstract user interface model which incorporates abstract context objects; At this design step the actual (concrete) context is not known so we will model the context in an abstract way using Abstract context objects, here the task model is not a single tree, set of trees each represents a single context, by filtering the tasks according to the context they are supported by, for example a task showing a map will be removed for abstract context with a device with low resolution, or a user with disability.

Step 2: For every possible task tree calculating the abstract dialog model which finds out the set of presentation units that can be presented to the user at the same time then calculating the transitions between them , this step will find out multiple dialog models each model represents a context of use, Two types of transitions will be calculated:

1. Intra dialog transitions: transitions between presentation sets for one context of use.
2. Inter dialog transitions: transition from a dialog model of one context of use to another dialog of a different context of use. This model represents a transformation step from one context to another in the AMI environment and can be done by the designer to model all expected transition constraints by the user or the context change.

A method like finding the enabled task sets by Luyten et.al. [32] can be extended to find multiple dialogs, the current AUI in UsiXML does not support dynamic environment changes and calculated the AUI by mapping non leaf tasks of the task model to abstract containers and leaf tasks to abstract individual components giving nested so it restricts the navigation and presentation of the user interface according to task level in the tree, while in an AMI environment tasks may no longer belong to the same Abstract Container due to the fact that some of the tasks cannot be performed in the current(dynamic) situation of the task, consider three tasks t1,t2 and t3 they belong to the same Abstract Container and according to their algorithm(which uses Luyten's algorithm [32] for calculating presentation); if the user at run time is in an environment or the device he is using does not support the execution of task t1, then only t2 and t3 will be mapped to the same abstract container, unless they exchange information with t1.

Step 3: map the AUI to a CUI using set of transformations, this model is modality dependent.

Step 4: the CUI is reified by the final user interface by a rendering engine that customize the UI according to the toolkit on the target device, the final user interface

is also dynamic, and affected by context change as detected by the sensors.

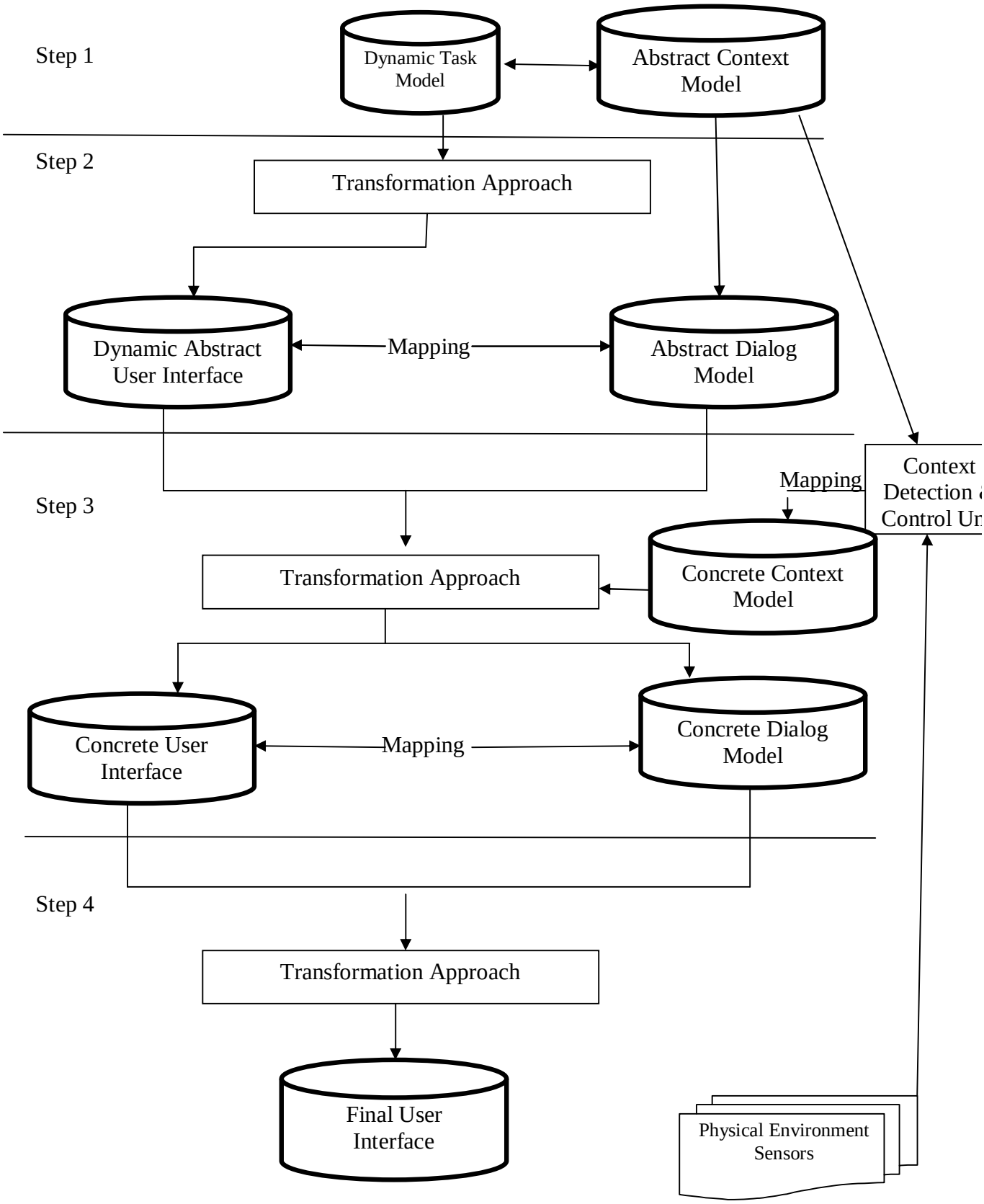

Figure 3: a design process supported by a dynamic environment model

A runtime architecture is needed to support the dynamic UI, at run time the change in environment is carried by the Context Control Unit (CCU), which gets the change of physical environment context from the sensors and according to the

current change will call for repeat the mapping of CCO to ACO, since the task model contains multiple trees, each represents an abstract context, the condition detected by CCU has to be fulfilled by an abstract context model, which selects the appropriate tree. This transformation will affect the navigation and accordingly the presentation of the user interface. So a recomputaion of the dialogue model will take place.
Notice that a dynamic dialog model is needed which automatically generates the dialog by step-wise reification process, this can be done by a method similar to the one in [31], once the concrete user interface has been specified the final user interface step 4, can be generated by compilation or interpretation of the UsiXML code to the target platform.

5. Conclusion

We have presented a framework and proposed design process and runtime architecture to support the creation of multi-device multi-modal and context-sensitive user interfaces. We believe this work can be an incentive for reconsidering the model-based user interface development approaches to enable the design of user interfaces in an AMI environment, specifically by changing the nature of models from static to dynamic.
The dynamic models can reflect the context change, and a step-wise reification is used to derive a final user interface.
The context change is controlled by the context control unit which abstracts the context, and a new reification process is followed to derive a new final interface suitable for the new context.
The approach presented is duly based on the clear separation of the context model and dialog from other components of the user interface. Such as separation presents several advantages such as it improves the readability of models and supports reuse of specifications also management of context change according to different design choices.
This method is clearly based on open standards like UsiXML which make it possible to assemble UI elements built with different tools.
Context switches as proposed can only affect the UI where the designer wants the UI to change by considering a fixed set of abstract contexts; a more general method is needed to identify context specification. Also suitable transformation system is needed to cope with the dynamic nature of the models.

References

[1] Puerta A. ,"A model-based interface development environment," IEEE Software., pp. 40–47, 1997.

[2] Coninx, K., Luyten, K., Vandervelpen, C., Van den Bergh, J.and Creemers, "Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems," Mobile HCI, volume 2795 of Lecture Notes in Computer Science, pp. 256–270, Springer, 2003.

[3] Mori G., Paterno F. and Santoro C., "Design and Development of Multi Device User Interfaces through Multiple Logical Descriptions," IEE Transactions on Software Engineering, Vol.30, No. 8, 2004.

[4] Clerckx *T.* Winters F. and Coninx K., "Tool Support for Designing Context Sensitive User Interfaces using a Model Based Approach," Proceedings of the 4th international workshop on Task models and diagrams TAMODIA '05, ACM Press, 2005.

[5] World Wide Web consortium, Useware Markup Language
http://www.uni-kl.de/pak/useML/

[6] Puerta, A., Eisenstein, J., *XIML: A Universal Language for User Interfaces.* 2001, RedWhale Software.

[7] World Wide Web consortium, Interface Specification Meta Language, http://decweb.bournemouth.ac.uk/staff/scrowle/ISML/

[8] World Wide Web consortium, User Interface Markup Language, http://uiml.org

[9] World Wide Web consortium, Renderer Independent Markup Language, http://www.consinsus-online.org

[10] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L. and López-Jaquero, V., USIXML: a Language Supporting Multi-Path Development of User Interfaces. *Proc. Of 9th IFIP Engineering Human Interaction and Interactive Systems*, 2004.

[11]TERESA:
http://giove.cnuce.cnr.it/teresa.html
Paterno, F., Santoro, C.: One Model, Many Interfaces. In *Proc .of the Fourth International Conference on Computer-Aided Design of User Interfaces*, pp. 143-154. Kluwer Academics Publishers, 2002.

[12] Stanciulescu, A., Limbourg, Q., Vanderdonckt, J., Michotte, B., Montero, F., A Transformational Approach for Multimodal Web User Interfaces based on UsiXML, Proc. of 7th Int. Conf. on Multimodal Interfaces ICMI'2005 (Trento, 4-6 October, 2005), ACM Press, New York, 2005, pp. 259-266.

[13] World Wide Web consortium, ConcurrentTaskTrees
http://giove.cnuce.cnr.it/ctte.html

[14] Montero, F., Víctor López Jaquero, V., Vanderdonckt, J., Gonzalez, P., Lozano, M.D., Limbourg, Q., Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML, Proc. of 12th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2005, Springer-Verlag, Berlin, 2005, pp. 161-172.

[15] Montero, F., Lozano, M.D., González, P., IdealXML: an Experience-Based Environment for User Interface Design and pattern manipulation, Technical report DIAB-05-01-4, University of Castilla-La Mancha, Albacete, 24 January 2005

[16] Michotte, B., Vanderdonckt, J., GrafiXML, A Multi-Target User Interface Builder based on UsiXML, Proc. of 4th International Conference on Autonomic and Autonomous Systems ICAS'2008 (Gosier, 16-21 March 2008), IEEE Computer Society Press, Los Alamitos, 2008.

[17] Szekely P., Luo P. and Neches R. ," Facilitating the Exploration of Interface Design Alternatives: the HUMANOID model of interface design," In CHI, pp. 507–515, 1992.

[18] Stirewalt K., "Automatic Generation of Interactive Systems from Declarative Models," PhD thesis, Georgia Institute of Technology, 1997.

[19] Szekely A. P., Sukaviriya N. P., Castells P.,Muthukumarasamy J. Salcher E., "Declarative Interface Models for User Interface Construction Tools: The MASTERMIND Approach," In EHCI, pp. 120–150, 1995.

[20] Stirewalt K. and Rugaber S. ,"Automating User-Interface Generation by Model Composition," In Proceedings of the IEEE International Conference on Automated Software Engineering , 1998.

[21] Francois B., Hennebert A., Leheureux J. and Vanderdonckt J, " Towards a Dynamic Strategy for Computer-Aided Visual Placement," In Workshop on Advanced Visual Interfaces, pp. 78–87. ACM press, 1994.

[22] Vanderdonckt J. and Bodart F., "Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In ACM Conference on Human Aspects in Computing Systems InterCHI' 93, pp. 424–429. Addison Wesley, 1993.

[23] Forbrig P. and Stary C., “From Task to Dialog: How Many and What Kind of Models do Developers Need,” CHI’98 workshop, 1998.

[24] Clerxkx, T.; Luyten K.; Conix, K.: DynaMo-AID: a Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Develoment, *Proc. EHCI-DSVIS’04*, p. 142-160, 2004.

[25] Stanciulescu, A., Vanderdonckt, J., Design Options for Multimodal Web Applications, Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2006 (Bucharest, 6-8 June 2006), Chapter 4, Springer-Verlag, Berlin, 2006, pp. 41-56.

[26] Stanciulescu, A., Limbourg, Q., Vanderdonckt, J., Michotte, B., Montero, F., A Transformational Approach for Multimodal Web User Interfaces based on UsiXML, Proc. of 7th Int. Conf. on Multimodal Interfaces ICMI'2005 (Trento, 4-6 October, 2005), ACM Press, New York, 2005, pp. 259-266.

[27] Vanderdonckt J. , Mendonca1 H., and Molina J. Distributed user Interfaces in ambient Envirinment, AMI 2007 workshop, Springer 2008 pp 121-130

[28] Vanderdonckt, J., Mendonça, H., Molina Massó, J.P., Distributed User Interfaces in Ambient Environment, Proc. of AmI-07 Workshop on “Model Driven Software Engineering for Ambient Intelligence Applications” MDA-AMI’07 (Darmstadt, November 7-10, 2007), Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2007, pp. 44

[29] Nikolaos Georgantas and Val´erie Issarny, User activity synthesis in ambient intelligence environments.

[30] Shankar Ponnekanti, Brian Lee, Armando Fox, Pat Hanrahan, and Terry Winograd. ICrafter: A Service Framework for Ubiquitous Computing Environments. In Ubicomp 2001: Ubiquitous Computing, Third International Conference Atlanta, Georgia, USA, September 30 - October 2, 2001, Proceedings, Lecture Notes in Computer Science, pages 56–75. Springer, 2001.

[31] Winckler M., Vanderdonck J., Stanciulescu A. and Trindade F., cascading dialog modeling with UsiXML, DSVIS 2008, LNCS 5136, pp. 121–135, 2008., Springer-Verlag Berlin Heidelberg 2008.

[32] Luyten, K., Clerckx, T., Coninx, K., Vanderdonckt, J.: Derivation of a Dialog Model from a Task Model by Activity Chain Extraction. In: Jorge, J.A., Jardim Nunes, N., Falcão e Cunha, J. (eds.) DSV-IS 2003. LNCS, vol. 2844, pp. 203–217. Springer, Heidelberg, 2003.