Task models and Interaction models in a Multiple User Interfaces generation process

Kinan Samaan, Franck Tarpin-Bernard
Laboratoire ICTT, Lyon
21, Av. Jean Capelle, 69621 Villeurbanne cedex - FRANCE
Kinan.samaan@insa-lyon.fr, franck.tarpin-bernard@insa-lyon.fr

ABSTRACT
In a Multiple User Interfaces (MUI) context, several models must be defined and adapted (tasks, user, domain…). Abstract models are progressively enriched in concrete models using pattern libraries and filtering processes. In this paper, we define the central role of the interaction model in MUI design and specification. This model manages the interaction between the user and the application, and ensures the link between task models, abstract interfaces and the functional core of the application. In our approach, we use AMF, a multi-agent and multi-facet architecture, to define the interaction model. We describe the structure and behavior of an AMF-based interactive system that provides multiple user interfaces.

Keywords
Multiple User Interface, design patterns, model-based, AMF, XML based language.

INTRODUCTION
In the last few years, a wide variety of computer devices emerged. It includes mobile phones, personal digital assistants (PDAs), Internet-enabled television (WebTV) and electronic whiteboards powered by high-end desktop machines. In each family, the number of variants can be huge (see phones). Today it is impossible and non economic to ask designers to manually adapt User Interface (UI).

Recently, many researches have been led and many approaches have been proposed to define a Multiple User

ntroller.

to us, the interaction model is one of the most important models to consider because, on the one hand, it manages the human-computer interaction, and on the other hand, it ensures the link with the functional core of the application. This model was often neglected in the steps of MUI generation. According to us, a tightly coupled relationship between the models is the only way to achieve automatic adaptation of applications at run-time.

TAMODIA'04, Prague, Czeck Republic.

Interface (MUI) [12]. The first approaches were based on description languages like UIML [1] and XIML (RedWhale) [20] and focus on web-based applications. These UIDL (User Interface Description Language) mainly organize adaptation processes in two levels or steps: an abstract level and a concrete level corresponding to the implementation in HTML, Java or WMT. Developers can use generic objects (i.e. button) and do not need to consider each specific implementation.

Then, model-based approaches appeared. Many of them rely on a task model. They filter a generic task model in order to define an abstract user interface and later build a concrete user interface. These approaches [8][15] try to increase the capabilities of traditional single-context task modelling to simultaneously support multiple contexts of use. In the CAMELEON project, Calvary et al. defined a unifying reference framework for multi-target user interfaces [3]. This framework tries to give a global view of the multiples approaches on MUI.

The improvement is important. However, interface models are generally static descriptions of interfaces with links to the functional core. As a consequence, they are very efficient for modelling basic interactions but are limited for modelling more sophisticated ones like "drag and drop". Currently they are dedicated to graphical interaction and need to be extended to manage multimodal and multi-style interactions. The natural way to do that is to include dialog controller components. Because we want to emphasis the role of interactions we call "interaction model" the model associated to the dialog co

According

The adaptation process forces the designer to clarify/explain the links between the task model, the interface model and interaction model. For the description and the adaptation of the interaction model, our approach relies on the AMF architecture (Agent Multi-Facets) [16]. Indeed, AMF have the following advantages:

- The multi-facets idea appears very interesting since there are multiple definitions of application presentations and abstractions.
- The possibility to define an abstract interaction model.
- The possibility to define patterns of interaction, packaging generic mini-models (sets of agents, facets, ports and administrators).

- The ability to insert at run-time these patterns in the abstract interaction model according to the context.

The goals of the AMF model are to help the design, implementation, use and maintenance of applications. Our approach consists in combining both a multiagent view (like PAC [4]) and a layered view (like Arch [2]). The multiagent view is used during the design and a layered technology is used for implementation.

Actually, agents are dual entities: one part located into the AMF engine manages the control of the interactions while another one, on the application side, manages widgets for interactivity and domain-dependent abstractions. Presentation classes and functional core classes are implementation classes written by the developers whereas AMF entities are manage by the AMF engine according to the AMF model that has been defined. The 5 levels of Arch model are present:

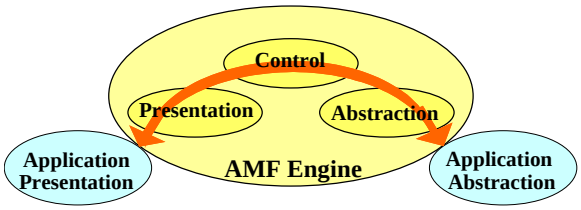

Figure 1 The Layers of the AMF implementation

Our General Approach of Multiple User Interface Design and Specification

To allow a more important variation at the interaction style level as well as at the implementation level, it is necessary to introduce a richer and generic description and to replace the language-based approach by an architecture-based approach [9][10]. In the AMF approach, we propose to start with a task model and to map it to an architecture-based abstract interaction model expressed in AMF, then to concretise this one in relation to the characteristics of the working platform. Once the concrete interaction have been chosen, the degrees of freedom available allow an ultimate adaptation to the user and the environment.

Most researchers consider that the generation of UI cannot be performed at runtime. Indeed, adaptation at runtime raises new challenges related to the continuity of interaction, especially for large applications or when context changes are detected.

However, like others [5] [19], we think that, thanks to the improvements of HCI description formalisms, we can build a MUI generation process where the designer specifies constraints before run-time and applications automatically adapt their own UI according to the context respecting the constraints.

Our approach consists in organizing the MUI generation process in 4 phases (figure 2):

- Abstract application definition phase,
- Interaction styles selection phase,
- Concrete interface generation phase,
- Final adaptation phase.

The first phase consists in modelling the generic task model and the abstract interface model, and defines the links between these two models and the abstract interaction model of the application. The designer builds these models once and for all. We will develop the contents of these models in the next section.

The second phase aims at dynamically generating the components of the interface that are adapted to the target. This phase is activated when a target (that is a triple < user, environment, platform > [17] like <“Michael”, “at home”, “TV”>) is running the software. The process consists in transforming the previous models with a first adaptation engine called “interaction styles selection engine”. For the adaptation, this engine considers two extra components: the platform model and a library of task and interaction patterns.

We can summarize the work of the adaptation engine in four points:

- It removes non-realizable tasks from the generic task model (e.g. removes a “print” task if the system does not detect a printer connected to the target). At this stage the engine also removes the elements of the abstract interface that are closely related to the removed tasks.
- According to the input devices of the target, the mechanism replaces each abstract task by a concrete task using a “task patterns” library (e.g. moving element with a pointing device).
- In parallel, the engine enriches the abstract interaction model by inserting the patterns that are associated to the task patterns using an “interaction patterns” library.
- The selection of patterns can be forced by constraints expressed by the designer during the design phase in addition to our own basic constraints that express ergonomics rules (guidelines table).

The third phase aims at generating a concrete interface where all the resources that will be used are selected but where the final parameters (layout, colour, volume…) are not set.

According to the characteristics of the devices (size, resolution, capacity…) and the user preferences, a second engine “concretization engine” selects among potential resources for each element of the semi-concrete interface, the ones that are more appropriate to the circumstances of use. The dependencies that have been defined between the domain objects are considered so that the choices are coherent.
In the final adaptation, the application and especially its interface take their final form. A third engine “finalization engine” builds the final layout of the interface, taking into account the presentation preferences described in the context model.

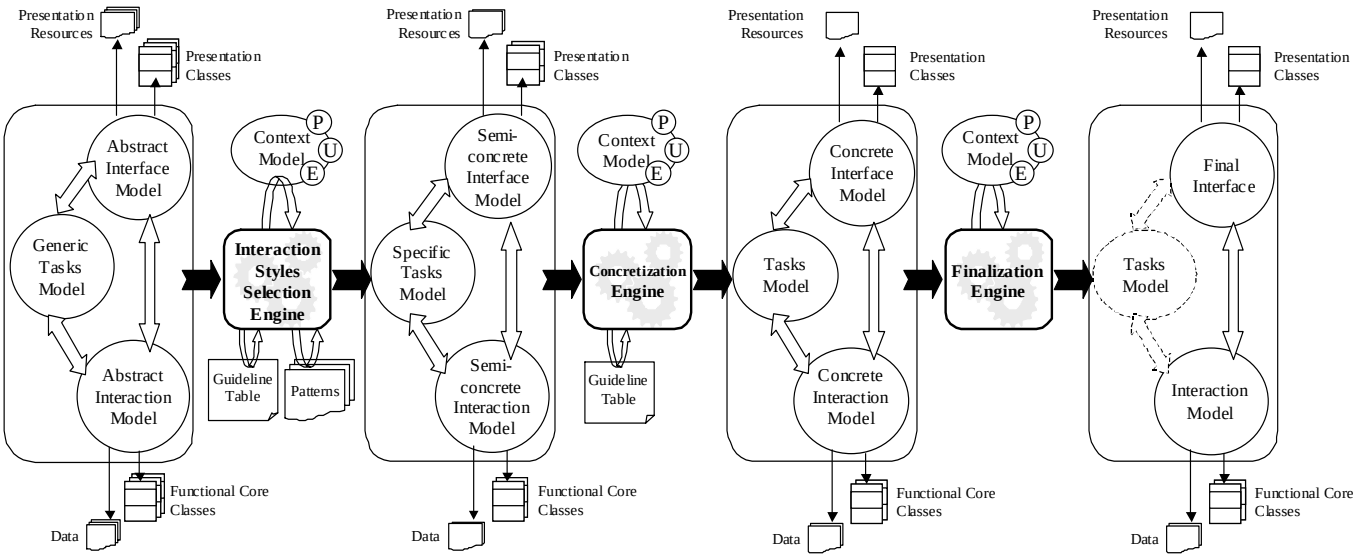

Figure 2 Our vision of the complete process of MUI generation

As we want to focus this paper on the links between the task model and the interaction model let's focus now on the core of our approach that is the AMF model.

BASIC NOTIONS OF AMF

AMF is a multi-agents and multi-facets architecture model that specifies the software architecture of an interactive application. It enables the design of reusable elements. It can be extended and adapted to the need of specific applications. The AMF model extends the PAC model and provides some new facets. As a consequence it is a hierarchical structure of reactive agents.

It provides a graphical formalism that represents the structure and specifies the temporal sequences of processes. Finally, a Java implementation of an AMF engine enables the execution of an AMF model coupled to applicative classes.

The class 'agent' is the basic component of AMF models. Each agent is made of facets and control administrators. It can imply other agents. Each class agent can generate several instances. Each facet incorporates logical communication ports and is associated to an applicative class where some functions, called «daemons», are mapped to the ports.

AMF proposes a unified formalism to model control components because such formalisms are rare and usually difficult to use in real contexts (see Petri nets for instance). Yet, these components are the major pieces of architectural models and it is of great importance to provide an efficient modelling tool. The control component of each agent is its main part because it manages all the communications between the facets of the agent and other agents. AMF defines 2 kinds of elements:

- At the facets level, communication ports present the services that are offered by the facet and the ones that are needed (respectively input and output ports).
- At the agent level, control administrators are connecting to communication ports. These administrators can easily be standardized (OR, AND, Sequence, etc.) and extended to handle complex controls such as multi-user synchronization or interaction tracking.

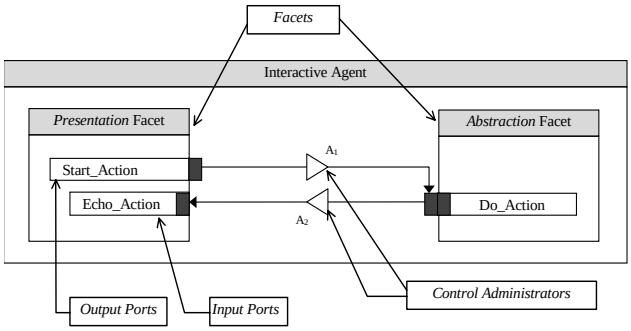

Figure 3 Basic elements of AMF architecture

In order to better understand the next figures, we briefly introduce two special features of the control administrators:

- After being activated, a target port is always returning a message to the source port. This "acknowledgement" message is generally ignored but it can be used to return data to the source port. When it is the case, the control administrator is represented with a black triangle (see figure 4a).
- The possible existence of multiple instances of a unique class drove us to provide a default mechanism that broadcasts messages from a control administrator to the target ports of all the instances of an agent. To be able to activate a specific instance of an agent, we add an optional parameter to the activated function in order to explicitly define a target agent. The identity of the agent is usually known only during runtime. So we do not need a new type of administrator but only a new activation technique. Yet, for a better understanding of the visual model, our advice is to add a little vertical

bar at the end of the administrator to explain that a filtering is done on the target agents (see figure 4b).

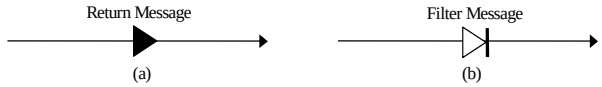

Figure 4. Return and Filter features of administrators.

The AMF Model can be published using an XML notation. Here is its Document Type Definition (DTD):

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Agent (Agent*, Administrator+, Facet+)>
<!ATTLIST Agent
    Name CDATA #REQUIRED
    Sub-agent CDATA #REQUIRED
    Type CDATA #REQUIRED
>
<!ELEMENT Administrator (Sources+, Targets+)>
<!ATTLIST Administrator
    Name CDATA #REQUIRED
    Type (Simple | Return | Filter | ReturnFilter | Sequence)
#REQUIRED
    TypeAC (Abstract | Concrete) #REQUIRED
>
<!ELEMENT Targets EMPTY>
<!ATTLIST Targets
    Name CDATA #REQUIRED
>
<!ELEMENT Sources EMPTY>
<!ATTLIST Sources
    Name CDATA #REQUIRED
>
<!ELEMENT Facet (Port+)>
<!ATTLIST Facet
    Name CDATA #REQUIRED
    Type CDATA #REQUIRED
>
<!ELEMENT Port EMPTY>
<!ATTLIST Port
    Name CDATA #IMPLIED
    Type CDATA #REQUIRED
    TypeIO (2 | i | o) #REQUIRED
    TypeAC (Abstract | Concrete) #REQUIRED
    DaemonName CDATA #REQUIRED
    FacetName CDATA #REQUIRED
>
```

Finally, a Java implementation of an AMF engine enables the execution of an AMF model coupled to application classes.

The AMF Engine

As we introduced it earlier, agents are dual entities: one part, located into the AMF engine, manages the control of the interactions while another one, on the application side, manages both widgets for interactivity and real domain-dependent abstractions.

To implement AMF architecture, we built an engine that manages all the AMF objects (agents, facets, ports and administrators) and their communications. The external elements, which are both objects that define the functional core of the application and objects that use a graphical toolkit, are linked to the AMF objects. For instance, each communication port is associated to a function called daemon in the Application side. This daemon is automatically triggered when the port is activated.

At runtime, a first agent is created. It analyses its XML description and consequently generate all its components (facets, ports, administrators) and recursively all its sub-agents. During the instantiation of facets, application classes (on the domain side or the UI side) are instantiated. As a consequence, if the XML description changes, then the application changes. We use this property in our MUI generation process: the goal of our adaptation engines is to build a version of XML description that fits in the context of use.

Then, for each user's action (button pressed, menu selection…), the corresponding event received by an application object (i.e. the one that manages the window) activates an output port of the associated AMF agent in the engine (↘ symbol in the graphical models). At the end of the control processing, input ports are activated and their daemons are run.

GENERIC TASK AND ABSTRACT INTERACTION MODEL OF THE APPLICATION

A task is defined as "*An activity that should be performed in order to reach a goal"* [6]. A Task Model describes the various tasks to be carried out by a user in interaction with an interactive system [15]. Various formalisms have been proposed to model the task model. We use the CTT notation and CTTE editor [11] for its description and modeling.

In our approach, the Generic task model contains regular nodes corresponding to common tasks and abstract nodes that will be "specialized" later in relation with the context of use. Figure 5 presents a Generic task model using a CTT notification of a gaming program called "Towers of Hanoi".

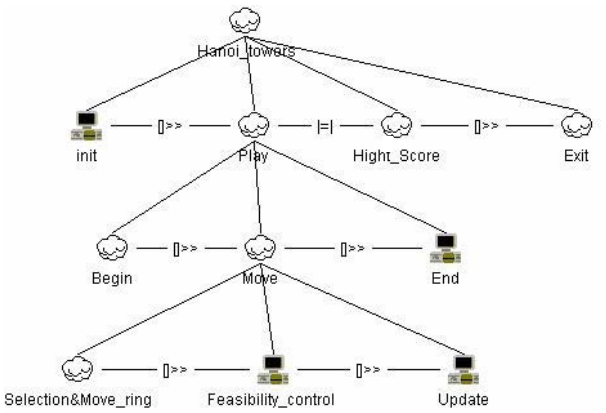

Figure 5 A Generic Task Model with some abstract tasks

This game consists in moving rings of different sizes. These rings are stacked up on three stems; they have an initial position and should be moved to reach a target-position. The shifting must respect the following rules: only one ring can be moved at a time, a ring with a given size cannot be placed upon a ring of a smaller size.

A "High score" feature has been added if the game is running on a connected platform.

Using the generic task model the designer can establish the abstract interaction model of the application. The

abstract interaction model is an AMF model that contains abstract ports related to the abstract tasks on the Generic Task Model. These ports represent functionalities that can be executed differently according to the specifications of the target. The abstract task is connected to abstract port in the interaction model (Figure 6).

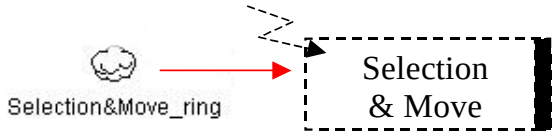

Figure 6. Relationship between the task model and the interaction model.

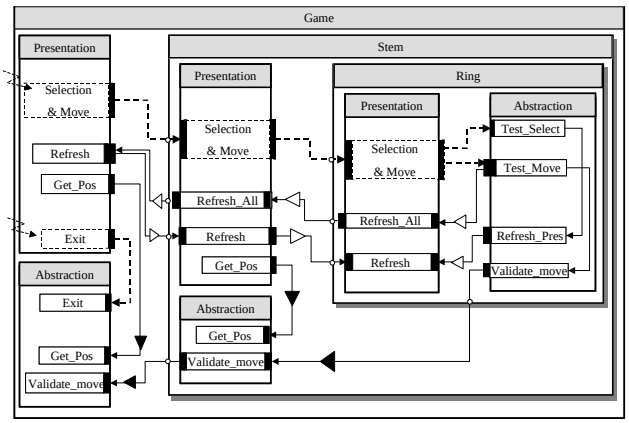

Figure 7 An Abstract Interaction Model with abstract ports.

The task of selecting and moving a ring in this model is an abstract task (elements are represented with dotted lines). Indeed, this action can be carried out differently according to the means of interaction that are available on the given target: a mouse, then the user may drag & drop a ring, or a keyboard, then he/she will type the figure of the source stem and after the number of the target. Figure 7 represents the complete abstract interaction model provided by to designer. Note that it also describes some semantics of the application through the definition of the agents (game, stem, ring).

To skip from an abstract interaction model to a concrete model, we need to replace abstract ports using patterns.

TASK AND INTERACTION PATTERNS

The AMF model is very adapted for implementing «design patterns» approaches because some combinations of agents - facets – ports constitute potential patterns that can be isolated and described.

The recent methods that have been designed for interactive software engineering emphasize on the task modeling, from macro-tasks to elementary tasks. The relations between task models and architecture are crucial for the design of efficient software, and this is even true in MUI software because it is necessary to define abstract interface before adapting it to the final interface.

As we will show it through an example, it is very easy to link parts of an AMF model to tasks. These parts are generally made of ports, administrators, facets and agents. As a consequence, it is easy to identify patterns of AMF model that are associated to a pattern of tasks. A pattern of tasks is a succession of sub-tasks that solves a specific problem in a particular context.

Thus, we have defined several patterns related to context of use and interaction means (mouse, keyboard…). Each pattern is used to achieve a specific kind of task in different contexts of use.

As an example we hereafter present two tasks and interaction patterns used for the selection and removal of an element among a set of elements located into a container. The first one is applied if the interaction is done with a keyboard (figure 8) and the second one is used when a mouse is available (figure 9).

Referring to Thevenin's works [18], we have first defined abstract patterns for elementary interactions:

- Selection (to select an item)
- Presentation (to display or to introduce with sounds…)
- Activation (to trigger an action)
- Specification (to input values)

Then, we concretize them according to various concrete interaction contexts and depending on our needs for our sample application. This leads for instance to sophisticated patterns like "Selection and Move".

The current list is obviously not comprehensive and has to be enriched. Designers themselves can easily extend it.

To help the pattern selection process, each pattern is qualified with a signature. This signature is a set of values that describes the interaction needs: screen space, sound device, pointing device… We plan to extend this signature with user-relative elements like cognitive load.

TASK AND INTERACTION MODELS ADAPTATION

Interaction patterns and task models fit together. The adaptation engine replaces the abstract task by a concrete pattern and the interaction pattern that is related to the task is inserted into the abstract interaction model. This replacement is done according to the characteristics of the target. Hence, concrete ports and connectors replace the abstract ports in the abstract interaction model.

If we consider the Towers of Hanoi example running on a platform with a mouse, the pattern presented in the figure 9 will be instanced. A **name-based rule** enables to maintain the link between the ports and the interface element that receives the action. Then, the task pattern replaces the abstract tasks in the task model of the application to produce the final task model of the application (figure 10). At the same time, the engine inserts the interaction pattern replacing the abstract ports in the interaction model. This process produces a final interaction model of the application (figure 11).

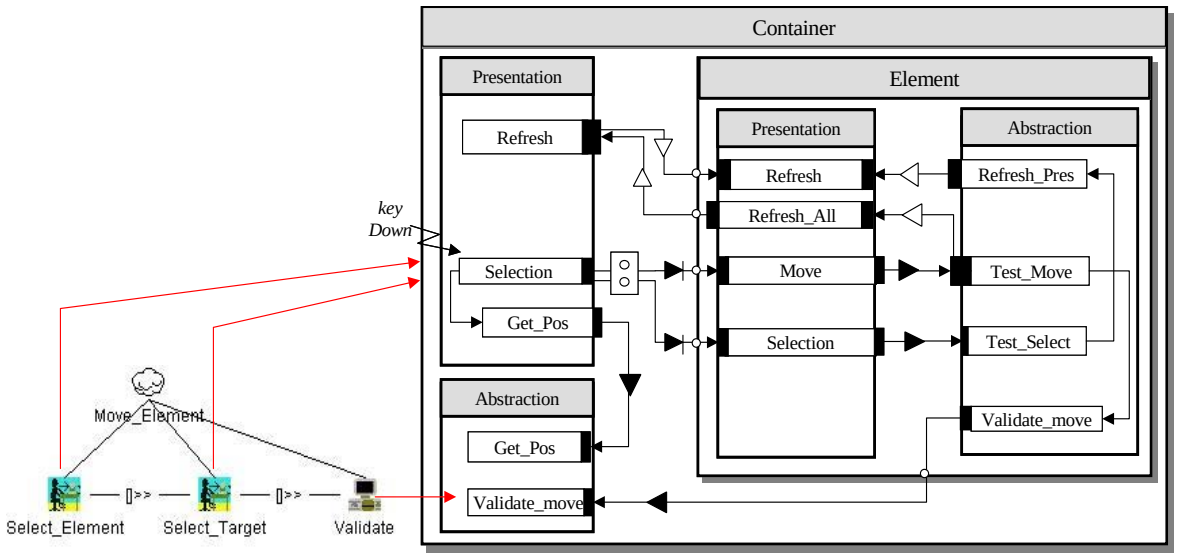

Figure 8 "Basic Select and Move" task and interaction pattern for a keyboard.

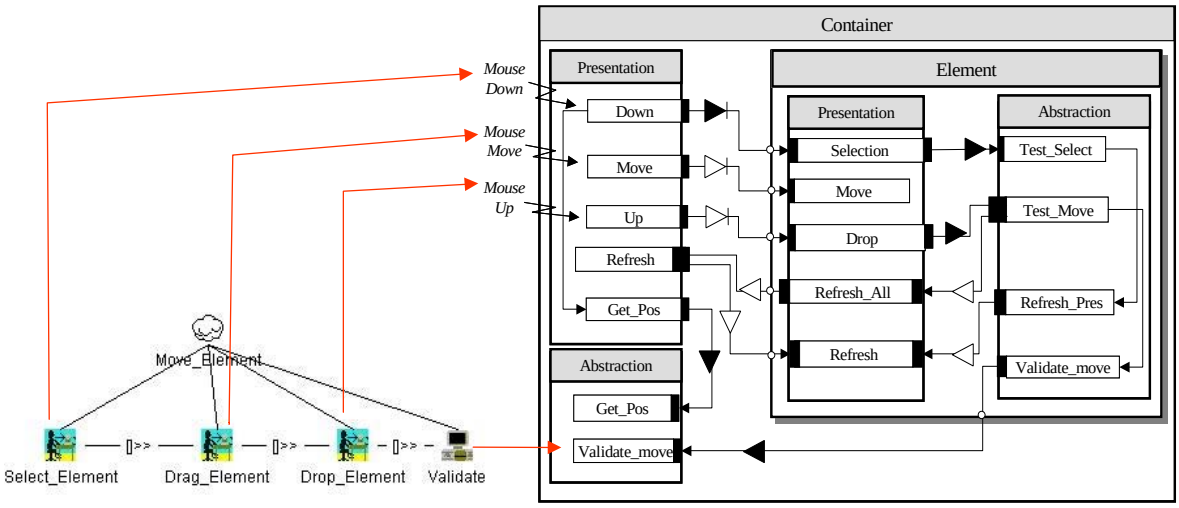

Figure 9 "Basic Select and Move" task and interaction pattern for a mouse.

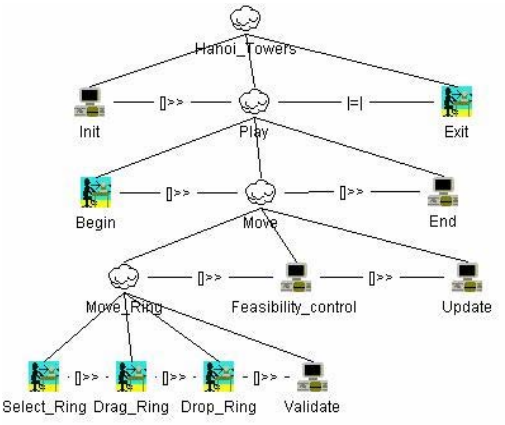

Figure 10 Concrete task model of the Towers of Hanoi

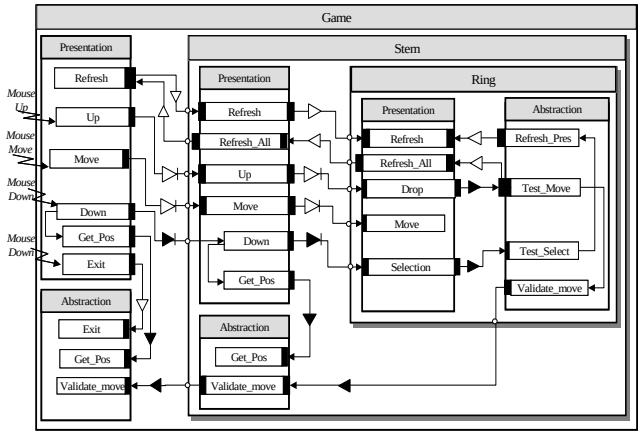

Figure 11 Concrete interaction model of the Towers of Hanoi with a mouse

The AMF engine executes the final interaction model, which is linked through the communication ports to the selected Java classes originally defined by the designer

(Presentation classes and functional core classes). These classes are using the presentation resources (images, sounds…) and the data resources (data base, web server address…) that have been selected by the previous process. See [13] for a detailed presentation of the relationships between the final AMF model and the application resources at run-time.

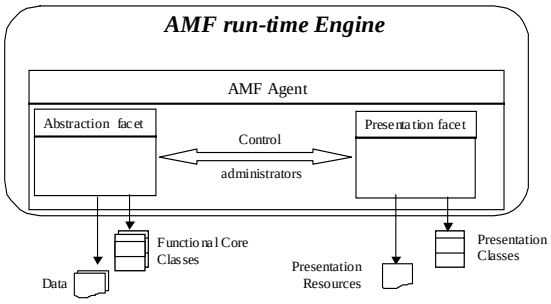

Figure 12 Relationships between the final AMF model and the application resources at run-time

CONCLUSIONS AND FURTHER WORK

In this paper we have briefly presented an approach that incorporates the interaction model in the complete Multiple User Interface design and specification process. To describe the interaction model, we use AMF, a multi-agent and multi-facet architecture. To adapt the abstract description of the application, we use task patterns and interaction patterns.

This approach is original in the sense that it tries to unify the task model, the interaction model and the resources of the application, i.e. the functional resources (Java classes of the application domain) and interaction resources (images, menus…).

We are aware of the difficulties and limits in considering the definition of a process that is wholly automatic. The complexity of the problem requires simplifications that inevitably lead to stereotyped and non-adapted interfaces to the specificity of materials. The introduction of the adapted task patterns and the interaction ones may decrease the complexity of the issue. However, it is obvious that the contribution of the designer should take place in this type of process. In this context, we will consider the introduction of a constraint definition file. The designer defines this file, which is used to restrict the modifications upon some elements during the process of the dynamic generation of the concrete interface.

Currently we use a waterfall approach to have dynamic adaptation to new contexts. To address migrations of context issues, it is necessary to be able to go back in the steps. Thanks to the properties of AMF (dynamic exchange of facets, insertion of agents, etc.) this should be possible but need to be deeply studied.

The current implementation includes the AMF engine and all the description formalisms necessary for our adaptation process. We are finalizing the first version of the adaptation engines.

REFERENCES

1. Abrams, M., Phanouriou, C., Baeongbaca,l A. L., Williams S. M., Shuster, J.E., UIML: An Appliance-Independent XML User Interface Language, In Computer Networks, Vol. 31, 1999, pp. 1695-1708.
2. Bass L., Pellegrino R., Reed S., Sheppard S., and Szczur M., “The Arch Model : Seeheim Revisited.” in CHI 91 User Interface Developper’s Workshop, 1991.
3. Calvary G., Coutaz J., Thevenin D., Limbourg Q., Bouillon L., Vanderdonckt J. A unifying reference framework for multi-target user interfaces, Journal of Interacting With Computer, Elsevier Science B.V, June, 2003, Vol 15/3, pp 289-308.
4. Coutaz J.: PAC, an Object Oriented Model for Dialog Design, in Proceedings Interact'87, North Holland, 1987, pp.431-436.
5. Coninx K., Luyten K., Vandervelpen C., Van den Bergh J., Creemers B. "Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems. In proceedings of MobileHCI 2003, Udine, Italy, 2003.
6. Gamma E., Helm R., Johnson R., Vlissides J. (1995), Design Patterns : Elements of Reusable Object-Oriented Software, Addison Wesley, Reading, MA.
7. Krasner G.E., Pope S.T. A Cookbook For Using the Model-View-Controller User Interface Paradigm in The Smalltalk-80 System. Journal of Object Oriented Programming, 1988, 1, 3, pp. 26-49.
8. Luyten K., Van Laerhoven T., Coninx K., Van Reeth F., «Runtime transformations for modal independent user interface migration». Interacting with Computers. Vol. 15, No. 3, June 2003. pp. 329–347.
9. Markopoulos P, Johnson P. Rowson J. Formal architectural abstractions for interactive software. International Journal of Human Computer Studies, Academic Press, (1998), 49, pp. 679-715.
10. Mori G., Paternò F., Santoro C. « Tool Support for Designing Nomadic Applications» Proceedings of IUI 2003, Miami, Florida, January 12-15, 2003.
11. Paternò F., Model-based Design and Evaluation of Interactive Applications. Springer-Verlag, November 1999.
12. Seffah, A., Radhakrishan T., Canals G. Workshop on Multiples User Interfaces over the Internet: Engineering and Applications Trends. IHM-HCI: French/British Conference on Human Computer Interaction, September 10-14, 2001, Lille, France.
13. Samaan K., Tarpin-Bernard F. « L’Utilisation de Patterns d’Interaction pour l’Adaptation d’IHM Multicibles ». IHM'03, CAEN-FRANCE, novembre 2003.
14. Samaan K., Tarpin-Bernard F. « The AMF Architecture in a Multiple User Interface Generation Process ». In Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages", Gallipoli, Italy, 2004, pp.71-78.
15. Souchon N., Limbourg Q., Vanderdonckt J. Task Modeling in Multiple Contexts of Use. In Proceedings of DSVIS’2002 Workshop. 2002.
16. Tarpin-Bernard F., David B.T., *AMF : un modèle d’architecture multi-agents multi-facettes.* Techniques et Sciences Informatiques. Hermès. Paris. Vol. 18. No. 5. p. 555-586. Mai 1999.

17. Thevenin, D., Coutaz, J. Plasticity of User Interfaces: Framework and Research Agenda. In Proceedings of INTERACT'99, 1999, pp. 110-117.

18. D.Thevenin : Adaptation en Interaction Homme-Machine : Le cas de la plasticité. PhD thesis, Université J.Fourier, Grenoble, France, déc.2001.

19. Vanderdonckt, J., Bodart, F., Encapsulating knowledge for intelligent automatic interaction objects selection. In: Ashlund, S., Mullet, K., Henderson, A., Hollnagel, E., White, T. (Eds.), Proceedings of the ACM Conference on Human Factors in Computing Systems InterCHI'93 Amsterdam, 24–29 April 1993), ACM Press, New York, pp. 424–429.

20. XIML Forum Site Web. http://www.ximl.org.