How to Exploit Abstract User Interfaces in MARIA

Fabio Paternò, Carmen Santoro, Lucio Davide Spano
CNR-ISTI, HIIS Laboratory
Via Moruzzi 1, 56124 Pisa, Italy
{fabio.paterno, carmen.santoro, lucio.davide.spano}@isti.cnr.it

ABSTRACT
In model-based approaches, Abstract User Interfaces enable the specification of interactive applications in a modality-independent manner, which is then often used for authoring multi-device interactive applications. In this paper we discuss two solutions for exploiting abstract UIs. We consider the MARIA language for such comparison. The overall aim is to improve the efficiency of the model-based process, thus making it easier to adopt and apply.

Author Keywords
Task models, abstract user interfaces, multi-device user interfaces

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI).

General Terms
Human Factors; Design.

INTRODUCTION

Model-based approaches for interactive applications have long been considered but their adoption so far has been limited. One of the most promising application areas is that of multi-device interactive applications in which they can hide the complexity derived from directly handling the various possible implementation languages. However, creating model-based descriptions of the interactive application for each target platform can still be tedious and requires time. It is thus important to be able to have one single model in the design process and then leave the refinements to the concrete descriptions for each target platform in the final part.

The CAMELEON Reference Framework [1] provides an indication of the various possible abstraction levels in model-based descriptions of interactive applications: Task Model, Abstract User Interface (AUI), Concrete User Interface (CUI), and Final Implementation.

In [2] the basic idea was to start from a single task model of a nomadic application and then derive the implementation of the user interfaces for the various target platforms through transformations into abstract and concrete user interfaces. We will call this as the "multi-AUIs approach" since it implies the development of a specific AUI for each target platform: all the AUIs so developed will be written using the same language/, although each of them provides a description of a different user interface that takes into account the features of each platform. This approach has then been adopted by other modeling languages, such as MARIA [3]. The authoring environment associated with MARIA (freely available at http://giove.isti.cnr.it/tools/MARIAE/home) supports the various abstraction levels considered by the CAMELEON Reference Framework through the ConcurTaskTrees (CTT) notation and MARIA itself.

Other works have included the consideration of Abstract User Interfaces in their approaches. In [4] the authors present a model-based approach able to generate device-specific WIMP UIs. Thus, they consider only graphical user interfaces, which need to adapt to various screen sizes. More specifically, this solution is based on a strong separation of concerns between UI structure and UI behavior. The starting point for the automatic generation process is the Communication Model, (which can be located at the Task abstraction level) which consists of Communicative Acts that can be assigned to either the user or the system. From a high-level Communication Model, a *device-dependent* UI model of the *structure* of a GUI (called "Structural UI Model") is directly generated. This is done to facilitate the optimisation process for a specific device (rather than building an AUI first and then refining it later for a specific device). Regarding the generation of the GUI *behaviour*, a *device- and modality- independent* UI Behaviour Model is also generated from the Communication Model. The *UI Behaviour Model* specifies the UI flow of interaction in a device-independent manner, namely in terms of the communicative acts that are received/sent at the same time, then forming a so-called "Presentation Unit". Then, the *UI Behaviour Model* is represented as a state machine having Presentation Units as its states, while the transitions between the Presentation Units are triggered by the Communicative Acts that are sent by the user. The generation of the Device-Specific *Structural UI Model* is done by using a declarative, rule-based, model transformation process. A *Structural UI*

Model basically consists of "Panels" connected each other through some relationships (e.g. choice, etc.).

Thus, since the resulting UI Behaviour Model and Structural UI Model are at two different levels of abstractions, they need to be "weaved" together in order to create a consistent, complete device-dependent UI model: the *Screen Model*, which consists of a (device-specific) Structural Screen Model and of a (device-specific) Behavioral Screen Model.

The *Screen Model* consists of a *Structural Screen Model* and of a *Behavioral Screen Model*, both at the concrete level. The *Structural Screen Model* specifies the concrete screens of the WIMP UI. The *Behavioral Screen Model* specifies the behaviour, i.e. the possible sequence of screens. Thus, the *Screen Model* provides a consistent specification of the WIMP UI at the concrete level. Each state in the Behaviour Screen Model corresponds to exactly one screen in the Structural Screen model and viceversa. The relations between the Structural Screen Model and the Behaviour Screen Model are represented by trigger events.

With respect to MARIA, the above approach presents the following differences. First that approach does not specify the *structure* of the UI at the abstract level. Among the reasons leading to this decision is the fact that knowing the structure of the UI at the abstract level is not that relevant because it is not possible to estimate the space needs of a screen at the AUI level (i.e. not knowing the widgets that will be finally used), then it skips this abstract level for the structural model. Instead, in MARIA, we still specify the AUI structure, which will be later refined into a device-dependent specification. In addition, in MARIA every model-transformation at one level produces models that are all consistently at the same level and therefore, there is no need to weave models together (in the previous approach, from a model at the task level they derive one model at the abstract level and another one at the concrete level).

A process for generating AUIs specified in USIXML starting from task and domain models has been recently described [5]. This process consists of a number of steps: i) first, the developer will link leaf tasks in the task model to the components appearing in the domain model; ii)then, information about the platform on which the generated UIs will run is specified (e.g. different platforms are assigned different weights by the UI designer); iii)then, tasks in the task model are assigned different weights based on task types (e.g. application, interaction,..); iv)tasks will be grouped together to create all the possible task combinations, and then some group of tasks which are judged unsuitable will be discarded; v)once the tasks have been grouped together and the platform has been specified, the system automatically generates the configurations suitable for this platform by selecting task groups previously created according to a number of criteria that can task-dependent or context-dependent (e.g. the task weight, the weight of task group, the device weight, the maximum weight of tasks for each platform); vi) finally, for each group, different AUIs are automatically generated (and stored in terms of USIXML specification), based on the configurations suitable for the specified platform and the mapping rules (the mapping rules identify abstract interactors by considering information in the task -and domain- models). Also the work by Tran et al. emphasizes the fact that on the one hand the AUI definition should remain independent of any platform/modality, while on the other hand the definition of suitable AUIs is generally done having already in mind the constraints imposed by the target platform. So, if the target platform is already decided, going through the AUI step could be no longer required and theoretically the step of defining an AUI could be skipped. However, for those authors it may still be interesting to identify all potential AUIs that could result from a task model and then decide by progressive refinement which ones could be the most suitable for a certain context of use.

With respect to the MARIA approach, one of the main differences is the fact that in [5] the designer cannot directly modify the automatically generated AUIs. This is done in order to preserve the consistency with the rules that have been applied to obtain these results. If the designer wants to change the AUIs s/he has to carry out the modifications at the task level. In the MARIA environment, the AUI can be directly modified by the designer and the modifications are taken into account in the following refinements (e.g. at the concrete level). Another difference of this work with our approach is the fact that in the transformation from ConcurTaskTrees (the notation used for task models) and MARIA we do not generate first all the possible AUIs as potential candidates and then refine the set. Rather, we try to converge to just one AUI through a transformation more complex than just mappings in which it is also possible to apply some heuristics in order to improve its results.

COMPARING TWO APPROACHES

In the following we discuss more in depth two solutions for exploiting AUIs within MARIA with the aim to identify the one that provides a more efficient support in developing multi-device user interfaces.

The Multi-AUIs Approach

In the original MARIA approach there are multiple versions of AUIs, one AUI for each specific platform considered. They can be obtained through direct editing, or by transforming the task model. Thus, designers have to use the same platform-independent concepts belonging to the same language to describe different user interfaces to take into account that some interactions and/or UI structuring are not suitable for some platforms. In the original MARIA a AUI was structured into several abstract UI *presentations*. Such presentations were aimed to group together the UI elements that are perceivable in the same time frame or that define a logical user interface unit. A number of

communication-oriented composition operators (grouping, relation, hierarchy, repetition) were also identified to specify from a structural point of view the relationships occurring between the abstract interactors within each Abstract Presentation. Moreover, it was also possible to navigate among the different presentations: this navigation was included as part of the dynamic behaviour associated to each single presentation and it was expressed through a number of *connections* between presentations.

In particular, the previous task model-to-AUI transformation considered the temporal operators in the CTT notation and calculated the so-called "Enabled Task Sets", groups of tasks that are enabled at the same time, and then it associated presentations to each of them. In order to avoid obtaining fragmented user interface descriptions various heuristics were defined for merging such Enabled Task Sets. For example, such heuristics indicate that if two sets share most elements then they can be merged, or if one set is composed of only one element then it can be merged with others, or if two sets differ for only one element and the different elements are connected through an enabling operator then they can be joined together. When targeting large screen interfaces the idea was to apply even more such merging of potential presentations in order to have fewer of them with more user interface elements in each presentation.

However, the multiple AUI specifications turned out as a burden for the designers that had to manage all of them. Therefore, another approach (the "Single-AUI approach", see next section) has been considered.

The Single-AUI Approach

It order to find a more effective way to specify AUIs a possibility is to remove the structuring at the *presentation* level (while still having composition elements and operators). The consequence is to have just *one specification* in which we dynamically describe, the elements that are perceivable at the same time (as before, this information can be derived from the task model). To some extent, removing the presentation concept can be seen as reflecting the evolution of various recent technologies. For example, Web applications are no longer structured into static pages but are more and more dynamic aggregations of contents derived from various sources.

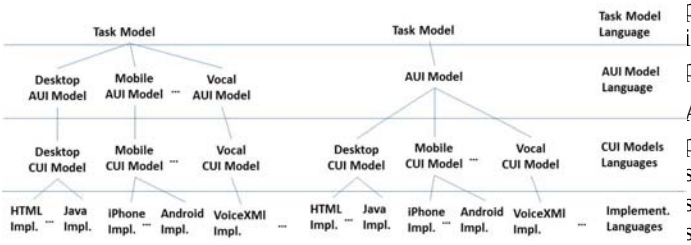

Figure 1: Comparison of the two methods.

Thus, in the new MARIA version we plan to derive a single AUI which can be derived from a task model specification or directly edited by the designer (see Figure 1).

However, removing the presentation concept has implications in the model transformations, in particular in the transformation from task models to AUIs. Now, to carry out the latter transformation we have to associate the Enabled Task Sets to interactor composition instead of presentations, and we need to derive some properties of such interactor composition by analysing the corresponding, underneath, group of tasks they support.

In addition, since the *presentation* concept as well as the associated *navigation* among presentations have disappeared, in the new approach we need other mechanism to dynamically specify the dynamic behaviour of the UI.

Therefore, in the Single-AUI approach, in order to fully specify a UI at an abstract level, we decided to associate various attributes to the various parts of an abstract UI specifying their relevant characteristics from both a *structural* and *dynamic* point of view.

Attributes concerning the *structural* organisation are those allowing –as before- to specify in which way the elements are statically organised in the groupings: e.g. whether there is an order among elements, whether there is a repetition, etc.

The properties relevant from a dynamic point of view will allow the specification of the dynamic behaviour of the UI by exploiting *event handler* specifications. The latter information should help in specifying the flow of interaction occurring in the UI, in a way independent from any device and modality. Then, each UI part can either have only structural attributes, or both structural and dynamic attributes.

An example of dynamic attribute is *perceivability*. It indicates whether or not the corresponding element (or composition of elements) is perceivable (or not) to the end user. The event handlers associated to interactors should indicate in their action part when to change the value of such attribute for any element. The resulting dynamic behaviour will then be appropriately rendered at the concrete (and also at the final) UI level through appropriate, platform- and modality-dependent techniques. This information could be included in a dialogue specification part in order to make it easier to access and interpret.

Another aspect that should be taken into account in the new proposal is the fact that in the single-AUI approach the single abstract UI specification will then be the common starting point for deriving the different concrete specifications for the various platforms. Therefore, the new AUI specification, yet unique, should be expressive enough to be used for deriving a refined, concrete UI for all the platforms considered. Therefore, in this single AUI

specification, we should still be able to identify the UI parts that are suitable for, or targeted to only some specific platform(s). In the previous approach the information about the platform was implicit, as we derived one AUI for each platform.

In the Single-AUI approach, each AUI elementary interactor or composition of interactors is *annotated* with information allowing to identify the suitable platform(s) associated to it (in case it is not suitable for all). This information can be derived from the nomadic task model in case it was first developed.

Another related issue is to identify the single AUI from which derive the various CUIs. In the case of transformation from task model, it can be automatically obtained from it, and having the perceivability attributes defined consistently with the dynamic behaviour defined by the temporal operators included in the task model. Indeed, the issue of avoiding fragmented user interface can still be addressed in a way similar to the other solution: heuristics similar to those used currently in MARIA could be applied.

In this case the merging would consider the composition interactor sets, the set of interactors associated with each composition operator.

This can be applied in the Task-to-Aui transformation but also in the Aui-to-Cui transformation. In the latter case the merging should take into account the features of the target platform and this does not necessarily mean that they become member of the same group but that they are perceivable at the same time.

Possible driving criteria in this process can be the number of interactors that would be perceivable at the same time or an estimation of the expected cost in terms of presentation resources (e.g. screen size). Thus, depending on the target platform we can set limits for these criteria to the number of interactors that can be perceivable at the same time and then merging process can be applied until such limit is reached.

ADVANTAGES AND DISADVANTAGES OF THE SINGLE-AUI APPROACH

If we analyse the two processes (represented in Figure 1) we can identify some advantages of the new approach.

Advantages of the Single-AUI approach

- Only one, self-contained Abstract UI specification is held, instead of multiple AUI specifications;
- Only one transformation is needed to move from the task model level to abstract UI level; this results in a gain in efficiency in the transformation process
- No more replication of UI elements within the UI specification appears (there is no more need to replicate UI elements in different UI presentations), thus making the specification more compact.

Disadvantages of the Single-AUI approach

- Less intuitive specification and visualisation of AUIs (and CUIs) since it is less intuitive to identify the elements that are perceivable at the same time with one less element of structuring;
- Both the transformations from AUI to CUI and from CUI to FUI should be adapted to suit such changes

AN EXAMPLE APPLICATION

In order to clarify the discussion we can consider the example in [6] in which two different AUIs are obtained for desktop and mobile platform. They are structured into three (mobile platform) and one (desktop platform) presentations.

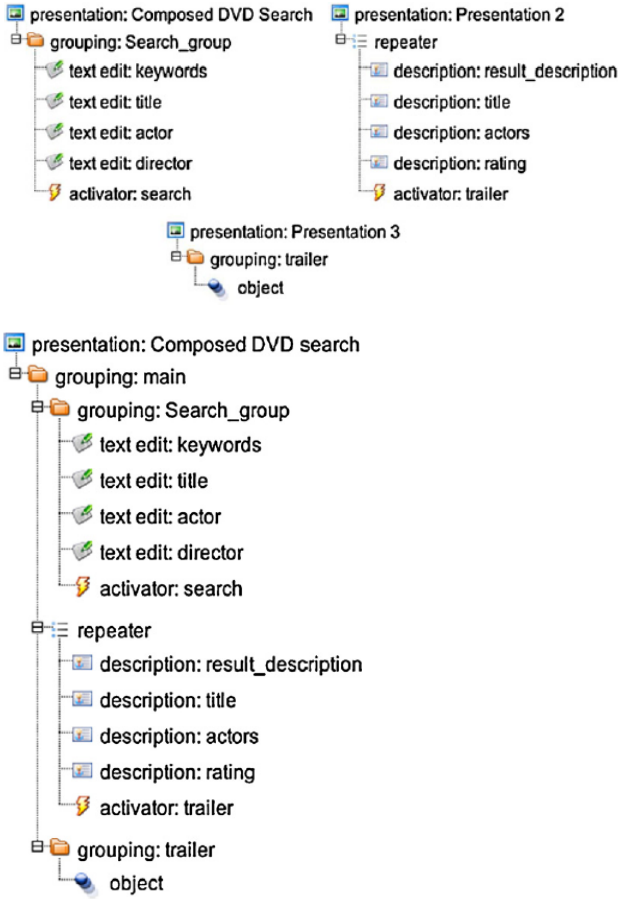

Figure 2: An example of Multi-AUIs approach.

In the case of the Single-AUI approach we would have a single AUI corresponding to the main grouping of the single presentation. In this case through the perceivability attribute it would be possible to differentiate the dynamic behaviour of the two user interfaces in the corresponding concrete user interfaces. For example, both could share that when the search is activated then the group associated with the search results should become perceivable. However, in the case of the mobile interface when the search is activated then the group of input parameters should set its

perceivability to false in order to gain further screen space for showing the results.

In the end the two approaches can generate similar results in terms of user interfaces. The potential interesting difference is in the efficiency of the process, since the Single-AUI approach requires creating explicitly only one single abstraction model either directly or through the task model. However, in order to be effective this approach needs more intelligent and difficult to implement abstract-to-concrete transformations able to change the *perceivability* of the various compositions, also taking into account the features of the target platform. Indeed, in the Multi-AUIs approach such transformation is rather simple to obtain since it mainly consists in adding concrete attributes to the various abstract specifications.

CONCLUSION

In this paper we present our first thoughts on how to more effectively exploit Abstract User Interfaces in the development of multi-device interactive application. The idea is moving from a multiple-AUI approach to a single-AUI approach which should also better reflects the most recent technological trends. Some advantages and disadvantages have been also discussed in the paper.

REFERENCES

1. Calvary, G., et al., 2003. The CAMELEON reference framework. Deliverable 1.1, CAMELEON project. Available from: http://giove.isti.cnr.it/projects/cameleon/deliverable1_1.html
2. Paternò, F., Santoro, C., Spano, L.D.: MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans. Comput.-Hum. Interact. 16(4): (2009)
3. F.Paternò, C.Santoro, One Model, Many Interfaces, Proceedings Fourth International Conference on Computer-Aided Design of User Interfaces, pp. 143-154, Kluwer Academics Publishers, Valenciennes, May 2002.
4. Raneburger, D., Popp, R., Kaindl, H., Falb, H., and Ertl, D., Automated generation of device-specific WIMP UIs: weaving of structural and behavioral models. In Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems (EICS '11). ACM, New York, NY, USA, 41-46.
5. Tran, V., Vanderdonckt, J., Tesoriero, R., and Beuvens, F.. 2012. Systematic generation of abstract user interfaces. In Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems (EICS '12). ACM, New York, NY, USA, 101-110.
6. Paternò F. Santoro C. Spano L.D., Engineering the authoring of usable service front ends. The Journal of Systems and Software. Elsevier, Volume 84, Issue 10, October 2011, pp. 1806-1822