Multi-Platform User Interface Development from a Single Task Model

Mir Farooq Ali
Department of Computer Science
Virginia Tech
660 McBryde Hall
Blacksburg, VA 24061, USA
+1-540-231-3986
mfali@cs.vt.edu

Manuel Pérez-Quinoñes
Department of Computer Science
Virginia Tech
660 McBryde Hall
Blacksburg, VA 24061, USA
+1-540-231-2646
perez@cs.vt.edu

ABSTRACT
In this paper, we present an approach for building user interfaces (UI) for multiple platforms from a single task model representation. We use the Concurrent Task Tree (CTT) notation for the task model and the User Interface Markup Language (UIML) as the underlying implementation language for the user interfaces.

KEYWORDS: Task Model, Multi-Platform User Interfaces, UIML.

INTRODUCTION
New computing devices are continuously emerging that have differing interaction styles, input/output techniques, modalities, characteristics, and contexts of use. This has necessitated development of new techniques for building UIs for these devices. Some of the problems faced in building UIs for these different devices and platforms are the different layout features and screen sizes associated with each platform and device [2]. In addition, most of these new devices have different development languages, each with its own widget set and interaction metaphor, that further exacerbates the problem. For example, the Wireless Markup Language (WML) that is used to develop UIs for mobile phones uses the metaphor of a deck of cards while VoiceXML uses a conversational metaphor.

We use the User Interface Markup Language (UIML) as the underlying language for building UIs [1, 6]. A UIML interface contains separate `structure`, `style`, `content` and `behavior` sections. Details about the UIML language and its shortcomings with respect to multi-platform UI development can be found elsewhere [1, 2, 3, 6].

We use a task model as a single representation for the user interface that can be used to generate the UIs for different platforms. The task model yields generic UIML that is then used to generate platform specific UIML [2, 3]. The model-based user interface development community has extensively used the Task Model as an abstraction for building UIs for a single platform, including systems like ADEPT [4] and MASTERMIND [7]. One of the recent notations for developing a task model is the Concurrent Task Tree (CTT) notation, developed by Fabio Paternò [5]. CTT provides notations for different categories of tasks and temporal operators between sibling tasks.

There are four *categories* of tasks defined in the CTT notation. These categories are *user*, *interaction*, *application* and *abstraction*. User tasks represent cognitive and perceptual tasks performed by the user. The user interacting directly with the system performs interaction tasks. Tasks like pressing a key or entering some text are interaction tasks. Additionally, there can be different *types* of tasks within a particular category. For example, an interaction task could be a selection type task. Application tasks are tasks performed by the system. These might be performed either in response to some interaction task done by the user or on their own volition. Abstraction tasks are complex tasks that could be a combination of user, interaction and application tasks and are not used to represent tasks at the lowest level in the task tree. In addition to the tasks, there are twelve temporal operators in the CTT notation [5].

GENERATION OF UIML FROM TASK MODEL
We use different aspects of the task model to generate different elements of the UIML for different targets.

Generation of structure
The task hierarchy in the task model, along with a set of developer-provided mappings, is used to generate the `structure` of the UIML UI. Each task in the task model is mapped to a `part` in the UIML that corresponds to widgets in the target platform. The developer, while building the task model, needs to provide three sets of

information.

1. Mapping for each task: Each task in the task model gets mapped to one or more UIML `parts`. This provides a structural correspondence between the task model and the generated UI. For example, an interaction task of type selection can be mapped to any one of the following types of lists: generic list, pull-down list, generic menu, etc. There are default mappings for each task type. However, the developer can change these to better customize the UI.
2. The containment hierarchy or grouping: The developer needs to specify whether the parent-child relationship between the tasks in the task model is also correspondingly maintained in the generated UIML. For example, consider a task that has three subtasks. In one scheme, the task and all of its subtasks could be in a single container in the generated UIML. In another scheme, the subtasks could all be represented in separate containers. The developer needs to provide this information too.
3. The desired navigation style: This depends to a certain extent on the containment hierarchy described in item 2 above. For example, if the developer specifies a containment scheme in which the subtasks are all in separate containers, then a menu-like navigation style might be desired.

The developer needs to provide this information for each of the desired target platforms. One reason is that different target platforms have different widget or `part` sets. In general, the generated `structure` for the desktop family will be similar to the hierarchy of the task model. However, it will be quite flat for the phone family (for WML and VoiceXML platforms). This is primarily a factor of the target platform.

Generation of behavior

The `behavior` section of UIML describes the events associated with the UI when the user interacts with it. It is enumerated through a set of conditions and actions. This is generated primarily from the temporal operators between the tasks. In UIML, a generic vocabulary comprises of parts along with associated properties and events. So the generated behavior and events will also depend on the type of mapping provided for each task in step 1 above.

In the earlier step of generating the `structure`, additional UIML `parts` are introduced in some cases depending on the navigation style and target family. Additional behavior needs to be generated for these parts too.

The developer does not specify the layout, style and other presentation-related information in this first step of the generation process, which yields generic UIML. This is provided in later steps to systematically build and customize the UI for each target platform.

CONCLUSIONS AND FUTURE WORK

We have presented a development process in which a single task model is used to generate UIs for multiple platforms through developer guidance. The same task model is used to generate the structure and behavior of the UIs for different platforms. We are in the process of implementing the UIML generation algorithms and plan to test it by developing a disparate set of UIs for different platforms. The success of this approach lies in the amount of time and effort that can be saved by the developer compared to creating the UIs separately. The next step is to incorporate this algorithm in an integrated development environment where the developer can use a life cycle based approach in building the UIs starting from a high level design, in this case the task model.

REFERENCES

1. Abrams, M., Phanouri, C., Batongbacal, A., and Shuster, J. UIML: An Appliance-independent XML User Interface Language. In *Eight International World Wide Web Conference: WWW8* (Toronto, Canada, 1999).
2. Ali, M. F., Pérez-Quinoñez, M. A., Abrams, M., and Shell, E. Building Multi-platform User Interfaces with UIML. In *Computer Aided Design of User Interfaces III (CADUI'2002)* (Valenciennes, France, 2002).
3. Ali, M. F., and Pérez-Quinoñez, M. A. Using Task Models to Generate Multi-platform User Interfaces while Ensuring Usability. In *Human Factors in Computing Systems: Extended Abstracts (CHI'2002)* (Minneapolis, USA, 2002), ACM Press.
4. Johnson, P., Johnson, H., and Wilson, S. Rapid Prototyping of User Interfaces driven by Task Models. In *Scenario-Based Design: Envisioning Work and Technology in System Development*, J. M. Carroll, Ed. John Wiley and Sons, Inc., 1995, pp. 209–246.
5. Paternò, F. *Model-Based Design and Evaluation of Interactive Applications*. Springer, 1999.
6. Phanouri, C. *UIML: An Appliance-Independent XML User Interface Language*. Ph.D. thesis, Virginia Tech, 2000.
7. Szekely, P., Sukaviriya, P. N., Castells, P., Muthukumarasamy, J., and Salcher, E. Declarative Interface Models for User Interface Construction Tools: The MASTERMIND Approach. In *6th IFIP Working Conference on Engineering for HCI* (USA, 1995).