Adapting Component-based User Interfaces at Runtime using Observers

Javier Criado[1], Luis Iribarne[1], Nicolás Padilla[1],
Javier Troya[2], and Antonio Vallecillo[2]

[1] Applied Computing Group, University of Almería, Spain
{javi.criado,luis.iribarne,npadilla}@ual.es
[2] GISUM/Atenea Research Group, University of Málaga, Spain
{javiertc,av}@lcc.uma.es

Abstract. Model-driven engineering (MDE) already plays a key role in Human-Computer Interaction for the automatic generation of end-user interfaces from their abstract and platform-independent specifications. Moreover, MDE techniques and tools are proving to be very useful for adapting at runtime the final user interfaces according to the current context properties: platform, user roles, component states, etc. In this paper we propose a mechanism to adapt user interfaces at runtime. These user interfaces will be (re)generated through the dynamic composition of user-interface software components, depending on the *observed* properties of the environment and of the components' behaviour.

Keywords: UI Adaptation, UI Composition, MDE, Observer, Trading

1 Introduction

Model-driven engineering (MDE) already plays a key role in user-interface design and user-interface development [11]. However, experience is showing that MDE can be even more effective for user-interface generation at runtime. Specifically, it is possible that different final user interfaces can be generated at runtime from the same *abstract* specifications, according to end-user context properties such as platform, user roles, component states, environmental conditions, etc. In this context, it is important to count on variability mechanisms that provide the appropriate levels of adaptability required to dynamically adapt user interfaces at runtime. Several authors have proposed solutions to tackle this problem. For example, in [12] the authors describe an example to adapt the user interface transformation at runtime through the selection of different types of rules. Recently, the work in [9] presents a framework for specifying user interfaces transformations through the definition of rules. Using a different approach, in [4] high-level modelling techniques and low-level programming techniques are combined to achieve the required plasticity of user interfaces. Interfaces are modelled through the composition of user interface components, which are described in terms of both abstract specifications and executable code.

A refactoring process of user interfaces is introduced in [10], which preserves the architectural definition and makes use of object-oriented programming to define the components and the selection logic. Another example that adds flexibility to the transformation rules of user interfaces is presented in [1]. It describes the transformation templates which specify the selection of user interface elements based on contextual requirements. Although all these works deal with user interface adaptation, none of them combines the dynamic user interface composition using transformations that are also adapted to the context at runtime.

Our research work also focuses on user interface adaptation at runtime. Here, user interfaces are described by means of Architectural Models that contain the specification of *abstract* user interface components [6, 7], which combined together provide the required user interface functionality. The *realization* of such software architecture is achieved by a trader [8] that looks in existing repositories of user-interface components for those that fulfil the requirements imposed by the architecture and selects the right set of components for the application. User-interface adaptation is achieved by changing the software architecture of the application, using standard model transformations languages and tools. Every time a new architecture is identified (normally due to changes in the user requirements or in the running environment), the trader finds again the suitable components that realize it. This defines a two-stage process for user-interface adaptation, consisting on a *transformation* phase that changes the architectural models that define the structure of the user-interface application, followed by a *regeneration* phase that populates the new architecture [5].

In this paper we introduce a new level of adaptation, provided by the use of *observer* objects that monitor the state and behaviour of the components that realize the user-interface architecture. Observers are not a new concept. For instance, the CAMELEON-RT framework [2] uses *observers* to collect system context information which is then used by an evolution engine. In [13] *observers* are in charge of calculating the QoS properties of the system elements by monitoring their progress, which can then be used in rules for system adaptation. In our case, we were inspired by this concept to establish UI adaptation rules from the UI component monitoring. In our proposal observers are used to trigger the model transformations that accomplish the adaptation process. A second, and more interesting, use of our observers is that they can be used to trigger a lower-level adaptation process whereby the global architecture does not need to be changed, but only one of its realizing components. For example, think of a video component whose output resolution drastically decreases due to a low network throughput. In this case a change in the architecture may not be needed if the component admits a change in its configuration that toggles to B/W display mode. Alternatively, another video component can be selected to replace it if no tuning is possible and the new component allows solving this problem.

The rest of the paper is organized as follows. Section 2 introduces our proposal for component-based user interface adaptation, and Section 3 explains the use of the *observer* elements. Finally, Section 4 presents some conclusions and outlines future work.

2 Component-based User Interface Adaptation

As explained above, user-interface adaptation is achieved by changing the software architecture of the application using a two-stage process: a transformation phase and a regeneration phase. The first phase was described in [5]. In this paper we focus on the second phase.

The *regeneration* phase takes as input an architectural model (AM), which is the abstract definition of the UI application. The architectural models are dynamically generated through the composition of UI components. This process is performed by two actors, called *SemanticTrader* and *UIManager*, respectively.

In the first place we suppose that the abstract definitions of the UI components reside in a component repository that is inspected by the *SemanticTrader* [8], which builds a runtime component model (RTM) from the abstract architectural model. The RTM is built in two steps. In the first step, the trading process generates all those configurations of "candidate" concrete components that can fulfil the architectural requirements. In the second step, the trader calculates the optimal configuration from the information given by abstract components in the architectural model and the information provided in the concrete component templates. This optimal configuration constitutes the RTM. From the RTM model, the *UIManager* process is responsible for generating the final user interface (that is showed to the user) by assembling the concrete software components recovered from the component repository.

This adaptation schema uses several abstraction levels, in a similar way to the *Model Driven Development of Advanced User Interfaces* (MDDAUI) proposal used in the Cameleon framework [3]: (a) *Tasks and concepts*, which represent the interaction models and abstract component models of the system; (b) *Abstract UIs*, which correspond to our architectural models; (c) *Concrete UIs*, which are the runtime component models, and (d) *Final UIs*, which are equivalent to the interfaces that are generated by the *UIManager* and showed to the user.

3 Using *Observers* in the adaptation schema

In our proposals observers are used to monitor the state of components in the runtime model (RTM). Not all user interface components can be monitored, since they are by default considered as "black box" components. Therefore, only those component properties explicitly modelled as "observable" in its associated component template will be able to be monitored—for instance noise level, throughput, jitter, number of lost packets, last interaction time, etc.

Observers are formally defined in a monitoring observer model (MOM) which is associated with the runtime component model. Figure 1 shows the observer and component metamodel. An observer model (`ObserverModel`) contains elements of type `Observer`. An `Observer` element has a reference to the monitored component (`observed_component`) and contains elements of type `ObservedProperty` referring to those properties that are being monitored and belong to a concrete component of the component model at run-time.

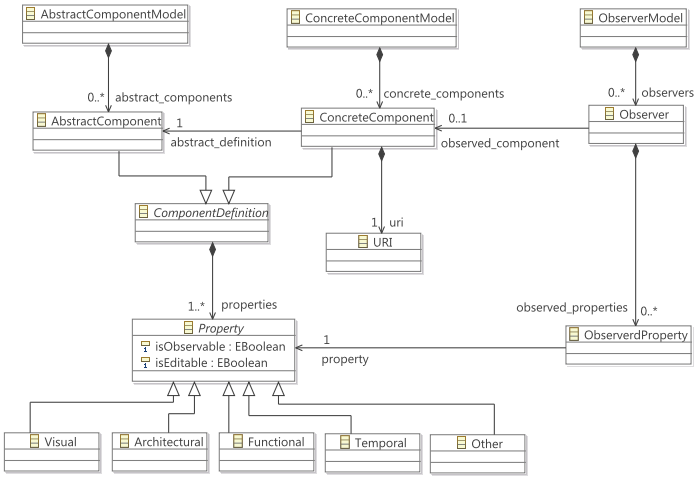

Fig. 1. The observer and component metamodel.

Once we have defined an observer model associated to a concrete user interface (*i.e.*, the runtime component model), we propose a transformation schema that makes use of this new information to enable the adaptation of user interfaces without causing any transformation (or modification) in the abstract user interface (*i.e.*, our architectural model). This new process (*MonitoringTransformation*) is a MIMO (*Multiple Input and Multiple Output*) model-to-model transformation that inputs the monitoring observer model (*MOM*) and the runtime component model (*RTM*), and generates a new runtime component model and a new monitoring observer model in the output (Figure 2). This *MonitoringTransformation* generates a runtime component model that conforms to the specification of the previous architectural model. In addition, it updates the observer model according to those "new" concrete components included in the model and the values of the observed properties of those concrete components that remain in the model.

The new runtime component model obtained by the *MonitoringTransformation* will also be regenerated by the *UIManager* showing accordingly the updated UI. From this adaptation schema, a change in the UI may be due to two reasons: (*i*) because of a change in the architectural definition, or (*ii*) because of a change determined by monitoring of the system. In the first case, a change in the architectural model may be determined, for instance, by the hiding, removing or addition of a component due to the presence of a detected event or the need to achieve a task. The second one concerns to those UI transformations that do not involve a change in the component architecture. For example, if an audio component is providing communications between two users of the system, and an *observer* object records a high noise level, the monitoring transformation process will generate a new runtime component model by replacing the audio component for a concrete one that can carry out this role in an improved manner.

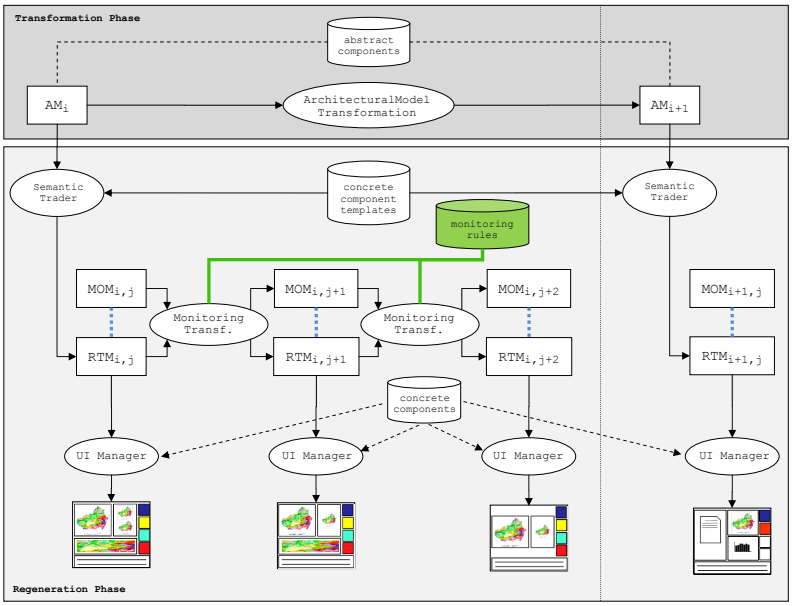

Fig. 2. The adaptation schema.

Regarding the Figure 2, we can observe those two kinds of adaptation. The right side shows the process when there is an architectural model change. This event causes an architectural model transformation that generates AM_{i+1} as output. On the other hand, the left side shows the adaptation process when there are no architectural model changes in AM_i. In this case, the runtime component model, regenerated by the *SemanticTrader*, takes part in the *MonitoringTransformation* which generates new RTM replacing the involved components.

4 Conclusions and future work

This paper has presented a user interface adaptation proposal at runtime, which extends our previous work [5, 7] with the use of observers that monitor the state of the user-interface components and of the overall system. Observers have provided us with an effective mechanism for triggering the changes in the high-level software architecture of the user interface application, and also for identifying and implementing low level changes that affect only to individual components.

This work is in an initial stage, much remains to be done. We now plan to build a concrete component repository that incorporates the new information about the component properties that can be observed and changed, using the metamodel presented in this paper. We will also need to update the *SemanticTrader* implementation [8] so that it works with the new repository. Finally, we need to develop all the machinery required to implement the models and transformations shown in Figure 2 in an automated way.

Acknowledgments. This work has been supported by the EU (FEDER) and the Spanish Ministry MICINN under grants TIN2010-15588, TRA2009-0309, TIN2008-00889-E and TIN2008-03107, and the Junta de Andalucía under grants TIC-6114 and P07-TIC-03184.

References

1. Aquino, N., Vanderdonckt, J., Pastor, O.: Transformation templates: adding flexibility to model-driven engineering of user interfaces. In: ACM Symposium on Applied Computing, pp. 1195–1202. ACM (2010)
2. Balme, L., Demeure, A., Barralon, N., Coutaz, J., Calvary, G.: Cameleon-rt: A software architecture reference model for distributed, migratable, and plastic user interfaces. Ambient Intelligence, pp. 291–302 (2004)
3. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A unifying reference framework for multi-target user interfaces. Interacting with Computers, 15(3):289–308 (2003)
4. Coutaz, J.: User interface plasticity: model driven engineering to the limit! In: 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems, pp. 1–8. ACM (2010)
5. Criado, J., Vicente-Chicote, C., Iribarne, L., Padilla, N.: A Model-Driven Approach to Graphical User Interface Runtime Adaptation. In: 5th Workshop on Models@run.time, CEUR-WS Vol 641, pp. 49-59 (2010)
6. Criado, J., Padilla, N., Iribarne, L., Asensio, J.: User Interface Composition with COTS-UI and Trading Approaches: Application for Web-Based Environmental Information Systems. CCIS 111, pp. 259-266, Springer-Verlag, Berlin (2010)
7. Iribarne, L., Padilla, N., Criado, J., Asensio, J., Ayala, R.: A Model Transformation Approach for Automatic Composition of COTS User Interfaces in Web-Based Information Systems. Information Systems Management, 27(3):207–216 (2010)
8. Iribarne, L., Troya, JM., Vallecillo, A.: A trading service for COTS components. The Computer Journal, 47(3):342 (2004)
9. López-Jaquero, V., Montero, F., González, P.: T:XML: A Tool Supporting User Interface Model Transformation. Model-Driven Development of Advanced User Interfaces, pp. 241–256 (2011)
10. Savidis, A., Stephanidis, C.: Software refactoring process for adaptive user-interface composition. In: 2nd ACM SIGCHI symposium on Engineering Interactive Computing Systems, pp. 19–28. ACM (2010)
11. Schaefer, R.: A survey on transformation tools for model based user interface development. Human-Computer Interaction. Interaction Design and Usability, pp. 1178–1187 (2007)
12. Sottet, J., Ganneau, V., Calvary, G., Coutaz, J., Demeure. A., Favre, J., Demumieux, R.: Model-driven adaptation for plastic user interfaces. In: 11th IFIP TC 13 Int. Conf. on Human-Computer Interaction, pp. 397–410. Springer-Verlag (2007)
13. Troya, J., Rivera, J., Vallecillo, A.: On the specification of non-functional properties of systems by observation. Models in Software Engineering, pp. 296–309 (2010)