# Utilizing Dynamic Executable Models for User Interface Development

Grzegorz Lehmann, Marco Blumendorf, Sebastian Feuerstack, Sahin Albayrak
DAI-Labor, TU-Berlin
Ernst-Reuter-Platz 7, D-10587 Berlin
firstname.lastname@DAI-Labor.de

**Abstract.** In this demonstration we present the Multi Access Service Platform (MASP), a model-based runtime architecture for user interface development based on the idea of dynamic executable models. Such models are self-contained and complete as they contain the static structure, the dynamic state information as well as the execution logic. Utilizing dynamic executable models allows us to implement a rapid prototyping approach and provide mechanisms for the extension of the UI modeling language of the MASP.

**Keywords:** human-computer interaction**,** model-based user interfaces, runtime interpretation

## 1. Introduction

Ambient environments comprising numerous networked interaction devices challenge interface developers to provide approaches that exploit these new capabilities. Multimodality, runtime context adaptation, personalization or even end-user development are examples for related challenges. We see two major features that can significantly empower user interface development. First is the possibility to modify the UI models at runtime. This feature allows to build self-adaptive user interfaces, which react to the current context of use. Furthermore it enables to close the gap between the design time and the runtime as the UI designer can alter the models of a running UI and see the results of his work immediately. The second feature is the possibility of extending the modeling language by extending its meta-models. This way the designer is no longer limited to one modeling language and is flexible enough to deal with challenges that will appear in the future. Moreover the architecture can then be customized for specific applications.

In this paper we present an approach based on executable models comprising static and dynamic information as well as the execution logic to form a foundation for the utilization of user interface models at runtime. In the next sections we present our realization of the Multi-Access Service Platform (MASP) using the Eclipse Modeling Framework (EMF). The demonstrated architecture allows the designer to work on models of a running application via model editors as well as the easy extension of the system (even at runtime) according to continuously changing requirements or end-user needs and preferences.

## 2. Multi Access Service Platform (MASP)

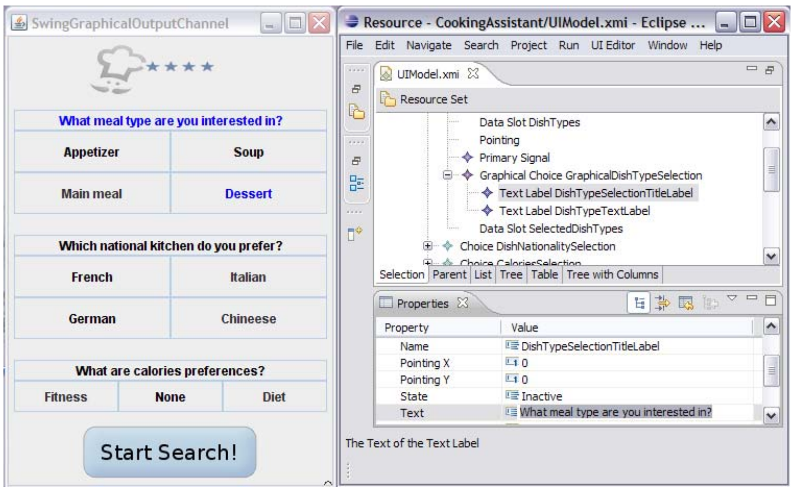
The MASP is a model-based runtime architecture which creates user interfaces from a set of models conforming to different meta-models. The user interface results from the execution of a model network comprising the task, domain, service, interaction and context models connected with customizable mappings [6]. The peculiarity of our approach lies within the executable nature of MASP's models and meta-models. In contrast to common static models, our executable models have a clearly defined execution logic and behavior specified in their meta-models. This makes them complete in the sense that they have "everything required to produce a desired functionality of a single problem domain" [5]. The executable meta-models provide the capabilities to express static elements as well as behavior and runtime evolution of the system in one single model. Additionally, the notion of the execution state as part of the model itself leads to models with an observable and manipulable state. Combining the initial state of a system, the processing logic, and the state information as part of dynamic executable models allows us to describe them as *models that provide a complete view of the system under study over time*. Thus, executable models run and have similar properties as program code. Other than code however, executable models provide a domain-specific level of abstraction which greatly simplifies the communication with the user or customer.

For our current implementation we have utilized the Eclipse Modeling Framework (EMF), which is a modeling and code generation framework integrated into the Eclipse IDE. EMF provides means to define meta-models, create models and appropriate editors. EMF is also capable of generating Java class structures for each meta-model and allows to express execution-defining meta-model elements in form of operations for which it then generates Java methods. These can afterwards be supplemented with Java code fragments which allowed us to define the execution logic inside our executable meta-models. One main feature of the Eclipse Modeling Framework supporting our runtime approach is the possibility to create model editors for the defined meta-models. We made use of this facility to build editors (Figure 1) that connect to the models of running MASP applications. This allows us to manipulate running applications and observe the effects of any changes immediately as they are instantly taken into account by the execution logic of the models.

## 3. Rapid Prototyping with Executable Models

We demonstrate the feasibility of our approach by showing the development process of an interactive recipe finder, allowing to search recipes, that match selected predefined criteria. When the search is completed the results are presented in form of a recipe list. The user may then either restart the search or select a recipe and proceed to its detailed description.

After defining basic model like the task model, providing the application workflow, and the domain model, providing access to application data, the application can already be executed because of the nature of the executable models. However, none of the models does provide a detailed description of the anticipated user


SwingGraphicalOutputChannel
What meal type are you interested in?
Appetizer
Soup
Main meal
Dessert
Which national kitchen do you prefer?
French
Italian
German
Chineese
What are calories preferences?
Fitness
None
Diet
Start Search!
Resource - CookingAssistant/UIModel.xmi - Eclipse ...
File Edit Navigate Search Project Run UI Editor Window Help
UIModel.xmi
Resource Set
Selection Parent List Tree Table Tree with Columns
Properties

Data Slot DishTypes
Pointing
Primary Signal
Graphical Choice GraphicalDishTypeSelection
Text Label DishTypeSelectionTitleLabel
Text Label DishTypeTextLabel
Data Slot SelectedDishTypes
Choice DishNationalitySelection
Choice CaloriesSelection

| Property | Value |
|---|---|
| Name | DishTypeSelectionTitleLabel |
| Pointing X | 0 |
| Pointing Y | 0 |
| State | Inactive |
| Text | What meal type are you interested in? |

The Text of the Text Label

**Figure 1:** Modifying an executable model of a running MASP application

interface or the interaction yet. We therefore proceed with the specification of the interaction model containing abstract and concrete user interface elements as proposed by the Cameleon reference framework [4]. While the application is already executed, we create elements allowing the user to provide the search criteria and start the search. The UI elements represent interactions for specific tasks and are therefore connected with tasks inside the task model by the means of mappings. While the models are connected with each other the UI starts to emerge on the display because the mappings synchronize the state of the UI elements with the current ETS of the task model. This way we can see the results of our work immediately. The UI elements representing the recipe search criteria are also connected to appropriate objects in the domain model, so that they appear on the screen immediately. Figure 1 shows the resulting (running) recipe finder UI on the left and the editor connected to the interaction model on the right.

In contrast to an earlier approach we presented in [1] the manipulations the UI designer performs happen directly to the model data structures, which are the same at runtime as well as at design time. Therefore the border between both becomes blurred. Being able to manipulate the models at runtime also paves the road for end-user development and self-adapting systems.

## 4. Extending the modeling language

In the second part of our demonstration we show the extensibility of the MASP by replacing one model with another one (conforming to a new meta-model) at runtime. We complete the definition of the interaction model with the creation of UI elements responsible for the navigation inside the Recipe Finder application. It should be possible for the user to navigate back and forth through the application, for example from the recipe details view either to the recipe list or to the initial search criteria configuration. As dialog modeling is not the responsibility of the task model a new model should be introduced. Therefore we will transform the task model into a state machine model and enrich it with additional transitions representing the desired dialog navigation. To achieve this we remove the mappings between the tasks and the UI elements and map the latter with states and transitions in the state machine model.

In order to achieve the described extensibility we have defined a meta-meta-model for the MASP. It distinguishes between definition-, situation- and execution elements of its executable models. A similar classification has also been identified by Breton and Bézivin [3]. The resulting structure of the meta-meta-model allows to generalize executable meta-models and relates them with each other by the means of mappings. As each executable model is an encapsulated entity on its own, to orchestrate multiple, independent models into one application we have also developed a special mapping meta-model. It enables the definition of custom mappings between elements conforming to different meta-models based on the structures given by the meta-meta-model. Providing an extra meta-model solely for mappings also allows to benefit from tool support and removes the problem of mappings hard-coded into the architecture, as has been already advised by Puerta and Eisenstein [6].

## 5. Summary and Outlook

In this paper we briefly described our approach of applying executable models to user interface development in order to enable the investigation of stateful models at runtime. The approach allows the inspection and manipulation of the application at runtime and provides easy extensibility of the utilized modeling language. The prototypical recipe finder application demonstrates the feasibility of our approach. In the near future, we want to further evaluate our approach, by implementing multiple models from different approaches like e.g. UsiXML[1] and we want to further investigate the implications of our approach to user interface development by creating enhanced UIs that facilitating context adaptation, self-awareness and self-adaptation.

## 6. References

1. Blumendorf, M., Feuerstack, S., and Albayrak, S. Multimodal User Interfaces for Smart Environments: The Multi-Access Service Platform. In *Advanced Visual Interfaces 2008*.
2. Blumendorf, M., Feuerstack, S., Lehmann, G., Albayrak, S.: Executable Models for Human-Computer Interaction, *submitted to DSV-IS 2008.*
3. Breton, E. and Bézivin, J. Towards an Understanding of Model Executability. In *FOIS '01.*
4. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Souchon, N., Bouillon, L., Florins, M., and Vanderdonckt, J.. Plasticity of User Interfaces: A Revised Reference Framework. In *TAMODIA 2002*.
5. Stephen J. Mellor. Agile MDA, June 2004.
6. Puerta, A.R. and Eisenstein, J.. Towards a General Computational Framework for Model-Based Interface Development Systems. In *Intelligent User Interfaces 1999.*

[1] www.usixml.org