# Dealing with Abstract Interaction Modelling in an MDE Development Process: a Pattern-based Approach

**Francisco Valverde, Ignacio Panach, Nathalie Aquino and Oscar Pastor**

Centro de Investigación en Métodos de Producción de Software
Universidad Politécnica de Valencia, Camino de Vera S/N, 46022 Valencia, Spain
{fvalverde, jpanach, naquino, opastor}@pros.upv.es

**Abstract** Currently, in the Model-Driven Engineering (MDE) community, there is not any standard model to define the interaction between the user and the software system. However, the Human-Computer Interaction (HCI) community has been recently dealing with this issue. A widely accepted proposal is the specification of the interaction at two levels or views: an Abstract Level, in which the User Interface (UI) is defined without taking into account any technological details, and a Concrete Level, in which the previous abstract models are extended with the information related to the target technology. The purpose of this chapter is to introduce the Abstract Level into the OO-Method MDE development process. Specifically, this chapter is focused on how the abstract interaction can be modelled by means of Abstract Interaction Patterns (AIPs). These patterns define a generic solution for an interaction between a user and an Information System (IS), without considering the technological details related to the final UI. In order to illustrate the approach, two AIPs are described.

## 1. Introduction

Model-Driven Engineering is considered to be a promising approach for the development of Information Systems (Schmidt 2006). Following this paradigm, the software application can be automatically generated by means of a Conceptual Model. From this model, several transformations are applied to obtain the final system implementation. Since this approach improves software development, several model-based methods have been proposed from both academic and industrial environments. However, these methods have been mainly designed to model the functionality and persistency of the IS (business and data-base logic, respectively) pushing into the background the UI modelling. Nowadays, the UI is gaining

enormous importance because the end-user can interact with software systems from a wide array of technological platforms (i.e. Desktop, Web, mobile devices, etc) which have different interaction mechanisms. Therefore, the interaction modelling must be considered a key requirement in a MDE development process.

The HCI community has been dealing with UI modelling for a long time. Previous HCI approaches have defined mechanisms to specify interaction such as ConcurTaskTrees (Paternò 2004) or UI description languages (Vanderdonckt et al. 2004). A widely accepted proposal is to define the UI specification at two main levels: an Abstract Level, in which the UI is modelled without taking into account platform or technological details and a Concrete Level, where the previous abstract models are extended with the information related to the target platform. However, in general terms, the HCI community has not taken into account how the interaction specification must be linked with the underlying system functionality. Therefore, with the proposals of the HCI community, a prototypical UI can be automatically generated, while a fully functional application cannot.

The main purpose of this work is to define a bridge between both HCI and MDE principles in order to improve how the interaction is defined at the Conceptual Level. Combining the best practices from both communities, a model-driven approach that defines both the interaction and the underlying IS logic can be obtained. In this chapter, an Abstract Interaction Model is introduced as the first step to achieving this goal. The basic elements proposed to define this model are the Abstract Interaction Patterns, which describe an interaction between the user and the IS at the Conceptual Level. The use of patterns provides the analyst with a common modelling framework to define generic interaction requirements. Furthermore, the knowledge represented by AIPs is general enough to be applied in different model-based software development methods. As an example of application, the OO-Method software production method (Pastor and Molina, 2007) has been chosen to illustrate how to include the Abstract Interaction Model on a MDE method. The main goal of this new Abstract Interaction Model is to improve the expressivity of the current OO-Method Presentation Model.

The rest of the chapter is structured as follows. Section 2 introduces the Abstract Interaction Model and describes two Abstract Interaction Patterns. Next, an overview of the OO-Method development process using the new Abstract Interaction Model is introduced. Section 4 compares the approach presented here with the related work. Finally, the concluding remarks are stated in section 5.

## 2. Defining the Abstract Interaction Model: Abstract Interaction Patterns

In the context of this work, the interaction is defined as the communication flows between the user and the IS by means of a software interface. The main goal of the Abstract Interaction Model is to describe the interaction without considering tech-

nological concepts of the UI. In order to achieve this goal, the interaction is represented by users and tasks. On the one hand, users represent a set of human subjects (i.e., Anonymous user, Customer, etc.) who share the same interactions with the IS. On the other hand, a task (i.e., 'Payment introduction', 'Log to the system', etc.) specifies an interaction with the IS to reach a goal in a specific software application. In addition, each user is related to a set of tasks that define its available interactions.

In order to describe the interaction represented by each task, this work introduces the concept of Abstract Interaction Pattern. An AIP defines a generic solution for a common interaction scenario between the user and the IS using a Conceptual Model. Instead of defining the interaction in terms of the final UI components involved, these patterns model the reason why the user interacts with the IS. Two common examples of interaction are the retrieval of data (i.e. show the user the cars that are available for rent) or the execution of a service (i.e., the payment of the car rental). To define these two interactions, the AIP models must be associated with the underlying models of the IS that represent the data and functionality. Hence, interactions are modelled over the data and functionality that the IS provides to the users.

To precisely define an interaction, the UI elements or widgets (i.e., buttons, forms, input boxes, etc.) that are used to interact must be specified. However, these widgets should not be included in the Abstract Interaction Model because they are technologically dependent on the target platform. To avoid this issue, a subset of the Canonical Abstract Components (Constantine 2003) is used at the Abstract Level to identify the interface components involved in the interaction defined by an AIP.

The advantages of using a structured and organized pattern language have been previously stated (van Welie 2003). Therefore, according to previous works on pattern language definition (Gamma et al. 1995, Molina et al. 2003), the AIPs have been defined using a pattern template with the following common sections: 1) A description of the *Problem* that the pattern is intended to solve; 2) The *Context* in which it is recommended to use the pattern; 3) A brief textual description of the proposed *Solution,* and 4) A real *Example* to show how the pattern can be applied. However, to apply these patterns in a model-driven development process, a more precise description is required. In particular, two main requirements must be satisfied:

1. The pattern should describe the entities and the properties of a Conceptual Model that abstracts the interaction. In the modelling step, that metamodel is instantiated as a model by the analyst to specify the required interaction in an application domain. In addition, the conceptual elements of that model are the source elements from which the model-to-code transformations are defined.
2. The pattern should include a precise description about the interaction expected in the final UI. This description must be used as a guideline to implement the

model-to-code transformation rules, which generate the expected UI code when the pattern is applied.

With the aim of addressing both requirements, two more sections are introduced to the pattern template:

1. *Metamodel*: This defines the concepts that the pattern abstracts by means of a metamodelling language. Additionally, this metamodel must include a clear description of the different entities, relationships, properties and constraints. The metamodel entities must also be related to the entities of the IS metamodel in order to establish the integration between interaction and functionality at the Conceptual Level. Therefore, the metamodel defines the static information used to create specific models of the pattern.
2. *Interaction Semantics*: This precisely specifies the interaction expected when the pattern is applied. This section shows the analyst how the different conceptual elements of the metamodel must be translated to implement the interaction defined. Therefore, it describes the Abstract Interface Components (Constantine 2003), the interface events and the business logic objects involved in the interaction. Additional interaction models, such as UML Interaction Diagrams, CTTs or Scenarios, are recommended to provide a better understanding of the semantics represented. To sum up, this section describes the interaction behaviour abstracted by the pattern.

One advantage of our proposal is that the concepts represented with AIPs are not coupled with a specific development method. As a consequence, these patterns can be used as guidelines to improve the interaction modelling in different model-based methods. Furthermore, new patterns detected in other approaches can be described as AIPs to promote reuse. To illustrate how this approach is applied, two AIPs are presented: the Service AIP and the Population List AIP. For reasons of brevity, a brief pattern summary is provided for sections one through four of the pattern template. In these examples, the *Essential MOF* (EMOF 2007) language has been used to build the pattern Metamodel, whereas CTTs have been used to define the Interaction Semantics as the JUST-UI approach proposes (Molina 2003).

## *2.1 Service AIP*

*Pattern summary:* This pattern abstracts the dialog between the User and the IS for the execution of an operation. That dialog can be subdivided into two basic interactions: the input of the operation argument values and the invocation of the operation. The interaction represented by this pattern can be applied in several tasks. For example, in an online rental service, the task “Create a new rental” can be defined using this pattern. The pattern application provides a form where the

user can enter the different argument values of the 'Rent' operation (the car name, the delivery and return date, etc.). When the user accepts the data entered (i.e., by clicking a button) the new rental is created in the IS.

*Metamodel:* A *Service AIP* (See Fig.1) is directly related to an operation and a set of *Input Arguments*. On the one hand, the *Input Argument* entity represents the input interface component in which the user must input the value for an operation argument. On the other hand, the operation represents a IS functionality that is composed of a set of *Arguments*. The value of these arguments is provided by the *Input Arguments* related to the *Service AIP*. Therefore, the *Service AIP* can be described as a view over a unique functionality provided by the IS. Finally, the *Launch Action* property of the *Service AIP* defines which interface component launches the operation execution.

*Interaction Semantics*: The first step of the interaction is the *Input of the Service Arguments* (See Fig.2). This task is decomposed into several '*Input Arg'* interactive tasks, according to the different *Input Arguments* defined in the model. The values can be inserted in any order as the operator (|||) means. When the required values have been introduced, the user should trigger an event interface (i.e., by clicking a button) to invoke the operation execution (*Launch Event* task). Finally, the *Execute Service* interactive task performs the operation with the values introduced in the previous tasks.

## *2.2 Population List AIP*

*Pattern summary:* Frequently, to avoid the incorrect input of a value, the interface provides a list of values from which the user must make a selection. The Population AIP represents the following type of interaction: the selection and input of a

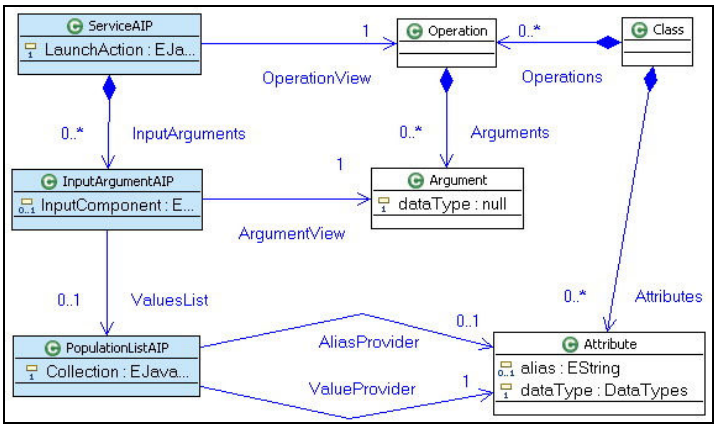


**Fig. 1.** Metamodels for the Service AIP and the Population List AIP

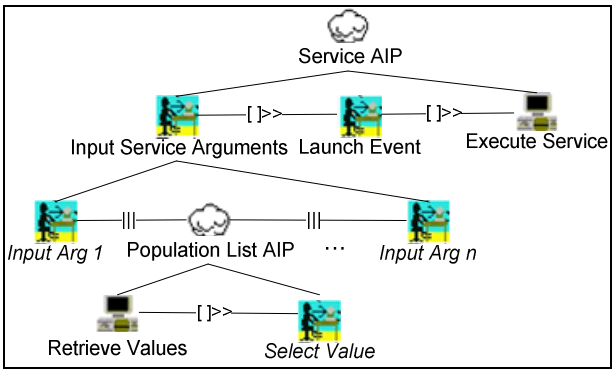


**Fig. 2.** ConcurTaskTree for the Service AIP and the Population List AIPs

value that has been previously retrieved from the IS. Taking into account the previous example, the 'Create a new rental' task, this pattern can be applied to the car name *Input Argument*. Hence, the interface provides all the cars available and the user only has to choose the desired value instead of typing it.

*Metamodel:* The Population List AIP can be associated to any metamodel entity that represents an input interaction. For instance, in Fig.1 a relationship with an *Input Argument* has been defined. The pattern provides the property *Collection* to represent the interface component that shows the user the collection of values. These values are provided by a class *Attribute* that is associated with the pattern by means of the relationship *Value Provider*. The relationship *Alias Provider* can be optionally defined to show an alternative representation of the value to be selected. For example, if the value to be selected is the car numeric id, an alias that shows the car description instead of the id will be more intuitive for the end-user.

*Interaction Semantics*: First, the application task '*Retrieve Values'* (Fig.2.) performs a query to the IS to retrieve the values of the *Value Provider* attribute defined in the *Class Attribute*. By default, these values are used to fill in the *Collection* property of the pattern, if there is not an *Alias Provider* defined. Finally, the interactive task '*Select Value'* represents the selection of the value by the user.

## 3. Introducing the Interaction Modelling in OO-Method

OO-Method (Pastor and Molina 2007) is an automatic code generation method that produces the equivalent software product from a conceptual specification. OO-Method provides a UML-based Platform-Independent Model, where the static and dynamic aspects of a system are captured by means of three complementary models: 1) The Object Model, which is defined as a UML Class Diagram; 2) The Dynamic Model, which is described as a UML Statechart Diagram, and 3) The Functional Model, which is specified using dynamic logic rules. Moreover, two

approaches have been traditionally proposed to support the presentation modelling in OO-Method: the JUST-UI approach (Molina et al. 2002) which has been industrially implemented in the OLINAVOVA tool (www.care-t.com), and the OOWS Web Engineering Method (Valverde et al. 2007) which provides Navigational and Presentational Models that are focused on the Web Application development domain. Concepts presented in both works have been taken into account to define the basis of the Abstract Interaction Model presented in this chapter. Therefore, the OO-Method Interaction Model is intended to be a unifying and extended proposal of the current Presentation Models.

The proposed OO-Method Interaction Model (see Fig 3.) describes the interaction using Conceptual Models (at the Problem Space) in order to automatically generate a Presentation Layer that represents the same interaction in programming code (at the Solution Space). Following the HCI principles, this Interaction Model is made up of two complementary views:

1. An Abstract View, which describes the different tasks that the user can perform with the IS in terms of AIPs introduced in the previous section. The AIP model entities are related to the OO-Method Object Model (the relationship *uses* in Fig. 3), which defines an interface with the data and the functionality from the IS. Hence, both models are integrated at the Conceptual Level.
2. A Concrete View defined as a set of Concrete Platform Interaction Patterns. These patterns are related to an AIP by means of a specialization relationship. Thus, their purpose is to extend the semantics of an abstract interaction with interaction concepts related to the target technological platform. In order to address concrete interaction requirements for different platforms, several Concrete Views can be defined (one per platform). The description of these patterns is beyond the scope of this chapter.

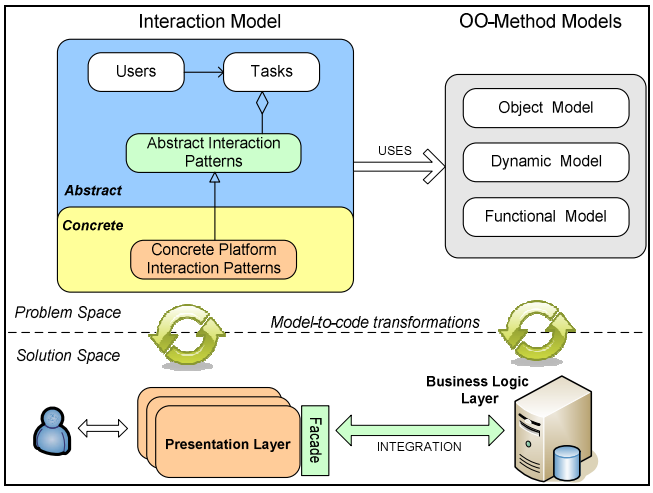


**Fig. 3.** Interaction Model in the OO-Method development process

Both views of the OO-Method Interaction Model are inputs of a model compiler, which according to a set of model-to-code transformation rules, produces the final Presentation Layer (See Fig.3). Additionally, the generation of the Business Logic Layer for several technological platforms has been industrially implemented in the context of the OO-Method development process. Therefore, a fully functional application, not just the UI, can be generated with the integration of the two code generation processes.

## 4. Related Work

In the HCI community, some proposals, such as USIXML (Vanderdonckt et al. 2004) and TERESA (Mori et al. 2004), have been made to model the interaction in an abstract way. In these works, the UI is designed independently of technological platform characteristics by means of ConcurTaskTrees (Paternò 2004). Taking into account the principles proposed by the Chamaleon Framework (Calvary et al. 2003), the UI is specified at different levels of abstraction. However, it is important to mention that both USIXML and TERESA can only represent the UI. Their abstract UI languages do not have the required expressiveness to generate the system functionality as OO-Method does. A recent approach for modelling the UI from the interaction perspective is the CIAM methodology (Giraldo et al. 2008). However this methodology is mainly focused on the definition of Groupware UIs.

In the literature, there are other proposals based on patterns to represent the interaction. In the work presented by Folmer et al. (2005), patterns are described as a bridge between the HCI and Software Engineering communities. Folmer et al. propose including patterns in the system architecture, but these patterns are not represented abstractly and, therefore, cannot be included in a model-driven development process. In the same line of reasoning, the work of Borchers (2000) introduces the advantages of using patterns to support Interaction Design: ease of communication among all team members and a common terminology to exchange ideas and knowledge. Borchers proposes formal descriptions of the patterns in order to make them less ambiguous, but those descriptions are also difficult to translate into Conceptual Models.

Furthermore, other works have proposed pattern libraries in the field of Interaction Design (Tidwell 2005, van Welie 2007). These patterns have been defined using real word examples from several applications. However, patterns are only described textually; guidelines or a proposed implementation are not provided to define a Conceptual Model to represent them abstractly.

Finally, the work presented by Sinnig et al. (2004) emphasizes the use of patterns as building blocks of different models (task, presentation and dialog) that define the interaction. However, these patterns are described using different XML languages and no pattern structure is proposed. As a consequence, these patterns

are difficult for analysts to understand. Also, this work does not describe, how to represent the Concrete View of the interaction.

The AIPs defined in this chapter not only address the description of how to represent the interaction abstractly but also how the pattern can be applied in a Model-Driven Environment. This approach extends the ideas proposed by Molina (2002) and provides a more precise description of the pattern metamodel and the interaction semantics involved. In addition, our approach can be associated with a Concrete View that improves the expressivity of the patterns taking into account the target platform characteristics.

## 5. Concluding Remarks

In this chapter, an approach for improving interaction modelling has been presented. As several works state (Molina 2002, Sinnig 2004), patterns are a recommended choice to represent the interaction in a model-driven development process. The presented AIPs provide a mechanism to promote the reuse of knowledge between different model-driven approaches and to guide the definition of the model-to-code generation process. The metamodel and the semantics that describe the interaction of the pattern are useful for integrating an AIP in different model-driven approaches. As a proof of concept, an Abstract Interaction Model that is related to the OO-Method approach has been defined.

A weakness of the approach is that there is no agreement on what the best model is to define the interaction semantics. Although in this chapter CTTs have been used, UML-based models may be more suitable to describe other AIP interaction semantics. Moreover, the introduction of the Concrete View may reveal the need for new conceptual primitives in the Abstract view.

It is important to mention two constraints of the proposal. First, this approach does not describe the aesthetic properties such as layout, fonts, colours, etc. Although it is true that characteristics of this type can have a great impact on the usability of the interaction, they should be addressed by another model. And second, the only modality of the interaction that is supported is carried out with common input/output devices.

Finally, further works will address the tool support required to define the Abstract Interaction Model in the OO-Method development process. As a previous step to reaching that goal, a metamodel of all the patterns must be developed together with the implementation of the corresponding model-to-code transformation rules. Furthermore, current work addresses how to define a Concrete view of the Interaction Model for modelling Rich Internet Applications.

**Acknowledgments** This work has been developed with the support of MEC under the project SESAMO TIN2007-62894

## References


Borchers JO (2000) A Pattern Approach to Interaction Design. ACM Conference on Designing Interactive Systems - DIS, New York, United States:369-378

Calvary G, Coutaz J, Thevenin D, et al. (2003) A Unifying Reference Framework for multi-target user interfaces. Interacting with Computers 15(3):289-308

Constantine L (2003) Canonical Abstract Prototypes for Abstract Visual and Interaction Design. 10th International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS), Madeira, Portugal:1-15

EMOF Meta Object Facility 2.0 Core Proposal (2007). http//www.omg.org/docs/ad/03-04-07.pdf. Accessed 29 July 2008

Folmer E, van Welie M, Bosch J (2005) Bridging patterns: An approach to bridge gaps between SE and HCI. Information and Software Technology 48(2):69-89

Gamma E, Helm R, Johnson R, Vlissides J (1995) Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, Boston

Giraldo WJ, Molina AI, Collazos CA, Ortega M, Redondo MA (2008) CIAT, A Model-Based Tool for designing Groupware User Interfaces using CIAM. Computer-Aided Design of User Interfaces VI, Albacete, Spain:201-213

Molina PJ, Melia S, Pastor O (2002) JUST-UI: A User Interface Specification Model. Proceedings of Computer Aided Design of User Interfaces, Valenciennes, France:63-74

Molina PJ (2003). Especificación de interfaz de usuario: de los requisitos a la generación automática. Valencia, PhD Thesis. Universidad Politécnica de Valencia.

Mori G, Paterno F, Santoro C (2004) Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. IEEE Transactions on Software Engineering 30(8):507-520.

Pastor O, Molina JC (2007) Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modelling. Springer, Germany.

Paternò, F (2004) ConcurTaskTrees: An Engineered Notation for Task Models. The Handbook of Task Analysis for Human-Computer Interaction. Lawrence Erlbaum Associates, United Kingdom:483-501.

Schmidt DC (2006) Model-driven Engineering. IEEE Computer 39:26-31.

Sinning D, Gaffar A, Reichart D, Forbrig P, Seffah A (2005) Patterns in Model-Based Engineering. In: Computer-Aided Design of User Interfaces IV. S Netherlands: 197-210.

Tidwell J (2005) Designing Interfaces. O'Reilly Media, United States

Valverde F, Valderas P, Fons J, Pastor O (2007) A MDA-Based Environment for Web Applications Development: From Conceptual Models to Code. 6th International Workshop on Web-Oriented Software Technologies, Como,Italy:164-178

Valverde F, Panach JI, Pastor Ó (2007) An Abstract Interaction Model for a MDA Software Production Method. Tutorials, posters, panels and industrial contributions at the 26th international conference on Conceptual modeling Auckland, New Zealand 83:109-114

Vanderdonckt J, Limbourg Q, Michotte B, Bouillon L, Trevisan D, Florins M (2004) USIXML: a User Interface Description Language for Specifying Multimodal User Interfaces. Proceedings of W3C Workshop on Multimodal Interaction, Sophia Antipolis, Greece:1-12

van Welie M, van der Veer GC (2003) Pattern Languages in Interaction Design: Structure and Organization. Ninth IFIP TC13 International Conference on Human-Computer Interaction. Zurich, Switzerland:527-534

van Welie M (2007) Patterns in Interaction Design. http:// welie.com. Accessed 29 July 2008