An Abstract Interaction Model for a MDA Software Production Method

Francisco Valverde[1], Ignacio Panach[1], Oscar Pastor[1]

[1]Department of Information Systems and Computation
Technical University of Valencia
Camino de Vera S/N, 46025 Valencia , Spain
Email:{fvalverde,jpanach,opastor}@dsic.upv.es

Abstract

Currently, most well-known model-based software production methods focus on defining the system functionality (business logic and persistence). However, the interaction between users and the system is too often not accurately described. Frequently an interface must be generated for multiple technological platforms (Desktop, Web, Mobile devices etc.) from the same model. The key issue is the model that was designed for describing a specific platform interface. When this model is used in other platforms, the final user interfaces have usability problems due to a lack of expressiveness at conceptual level. An interesting approach is to solve this problem from a MDA point of view. Two abstraction levels are defined in order to model interaction: a PIM (Platform Independent Model) or abstract level to describe interaction without taking into account technological issues and a PSM (Platform Specific Model) or concrete level to deal with platform concrete requirements. This paper explains in detail how the PIM level is defined in order to produce multiplatform user interfaces. This Abstract Interaction Model is made up of two models: an User Model that defines different types of users and an Abstract Interface Model to define the user interface. The final goal is to introduce these new models into OO-Method, an MDA-based software production method to produce software systems. As a result, a user interface which can be used as a prototype is automatically generated.

Keywords: Model Driven Development, MDA, Interaction modelling, User Interfaces, HCI

1 Introduction

An important topic to be analysed by the Software Engineering (SE) community is the interaction modelling. SE community has developed some well known methods to represent system structure and functionality in an abstract way, like the Class Diagram or the Sequence Diagram (UML 2003). However, SE community does not have a method widely used and accepted to represent the interaction between the user and the system.

Several methods provide an Interaction Model that only represents interfaces for a concrete platform like the Web. As a consequence, the migration process to another platform (i.e. mobile devices) is a difficult task. In addition, when this migration process is possible, the interaction model does not have enough expressiveness to describe specific characteristics of the new platform. Some authors have proposed user interface languages such as USIXML (Vanderdonckt 2004) or models such as UMLi (Silva 2003) to define interaction in an abstract way. But these approaches only define generic user interfaces and are not integrated in a full code generation process. Therefore, how the user interface is properly linked to the system functionality is not clear.

OO-Method is a software production method based on MDA (MDA 2003) that has a Presentation Model (Molina 2002) that represents the system user interface. For each modelling element, a software representation in several implementation languages is generated by a Model Compiler. Currently, the Model Compiler always produces the same code, from the same modelling element. This approach has a drawback because a modelling element could have several implementations, especially from a user interface perspective. For example, a user interface can be more adequate in a particular domain attending to usability constraints. The current OO-Method solution could be acceptable in a desktop environment where homogeneity between applications is recommended. However in web environments in which interfaces guidelines are less strict, the generated Web Applications may have different usability issues or do not meet customer requirements.

The main contribution of this poster is to propose an Abstract Interaction Model that redefines current OO-Method Presentation Model. This new definition, Interaction Model, emphasizes that interface is not only related to aesthetic aspects but it must take into account the communication between the user, the interface and the system. This new Interaction Model, to be compliant with the MDA development process proposed by OO-Method, is divided into two abstraction levels: An Abstract level (PIM), in which interaction is modelled without taking into account platform details and a Concrete level (PSM), in which specific concepts related to the target platform are defined. According to this approach, the set of modelling elements to be used in the PIM level must have the required expressiveness to represent multiplatform (Web,

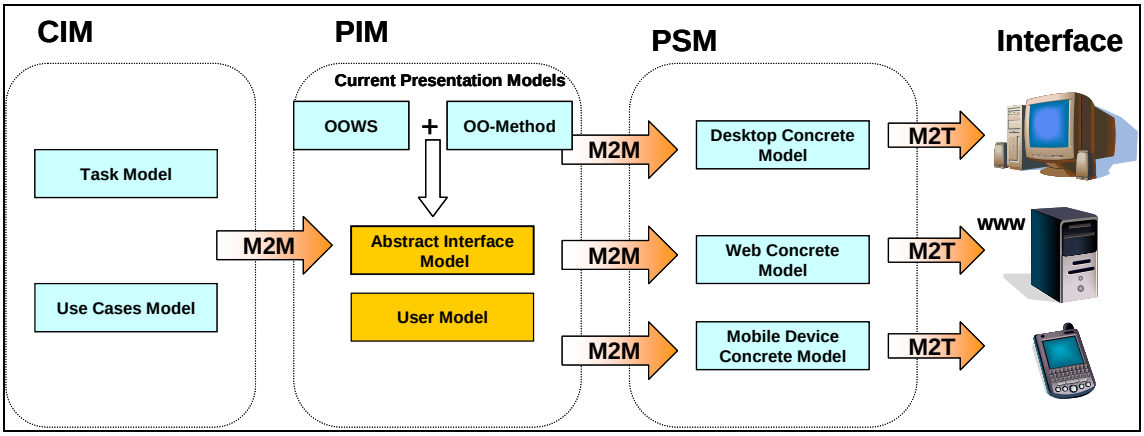

Figure 1: Abstract Interaction Model in a MDA development process

Desktop and PDA) aspects. Once the Abstract Interface Model is built, the Concrete Interface Model is generated by means of model-to-model transformations. At this level, analysts can refine the generated Concrete Interaction Model to introduce specific platform requirements. Subsequently, the Model Compiler transforms each Concrete modelling element to specific code. Therefore, the same Abstract Interaction Model can be used to produce a Web interface or a Desktop one.

This work is focused on the new Interaction Model defined at abstract level (PIM). The model is composed of two sub-models: the Users Model to represent different types of users that can interact with the system; and the Abstract Interface Model, that extends the current OO-Method Presentation Model with the interaction concept. Since the described modelling elements are defined independently from technological and methodological aspects, the same concepts can be used by another software engineering methods based on abstract models.

To accomplish the goals mentioned above, the paper is structured as follows: section 2 presents the OO-Method's Background and another works related to interaction. Section 3 describes the Interaction Model at the abstract level. Next, in section 4, a practical example that shows how an user interface is defined using the Abstract Interaction Model is presented. Finally, the conclusions and future research lines are stated.

2 Background and Related Work

Previous experiences of OO-Method and OOWS, our current methodologies, have been considered in order to define the new Abstract Interaction Model. OO-Method (Pastor 2001) is an Object Oriented software production Method that is MDA compliant (MDA 2003). OO-Method models the system in different abstraction levels, distinguishing between problem space (the most abstract level) and solution space (the lowest abstract level). The system is represented by a set of Conceptual Models: 1) A Class Diagram that represents the static structure; 2) The State and Functional Diagram that represents the behaviour; 3) The Presentation Model (Molina 2002), that is the current model used in OO-Method to describe abstract user interfaces. The Presentation Model is based on a pattern language that represents common interactions as information retrieval, service execution or data validation. The Abstract Interface Model described in this work, has been developed to be included in this method.

The industrial tool that supports OO-Method is called OlivaNOVA, (CARE) that has been developed in close cooperation with Care Technologies S.A. This tool produces functional systems for several platforms and implementation languages from an OO-Method Conceptual Model. However, users complained about the low usability of the generated web applications.

In order to solve OO-Method web usability issues, OOWS (Object-Oriented Web Solutions) was defined. OOWS (Fons 2003) is a web engineering method that provides methodological support for web application development. OOWS has been developed as an extension of OO-Method to support web-domain concepts. OOWS introduces the diagrams that are needed to capture web-based applications requirements, enriching the expressiveness of the OO-Method Conceptual Model. From a web engineering perspective, OOWS generates the code corresponding to the user interaction layer, and OlivaNOVA generates the business logic layer and the persistence layer. Some development process (Valverde 2007 and Giner 2007), are using OOWS to model and produce web interfaces.

In the HCI field, some proposals have appeared to model the interaction in an abstract way. Two proposals that share the same approach are USIXML (USer Interface eXtensible Markup Language) (Vanderdonckt 2004) and TERESA tool (Transformation Environment for inteRactivE Systems representAtions) (Mori 2004). In these works, the system is designed independently of technological platform characteristics. User interface specification is described in two levels: 1) Using a Task Model and a Concept Model in order to define an abstract model independently of the platform; 2) The Abstract Model is refined in a Concrete Model for a specific context of use. However, it is important to mention that both USIXML and TERESA are not an interface implementation language themselves. Their abstract user interface language does not generate system functionality as OO-Method proposes. Moreover, USIXML needs a transformation engine to interpret the model and to generate the interface code.

There are other proposals based on UML models, as WISDOM (Whitewater Interactive System Development with Objetc Models) (Nunes 2000), or UMLi (Da Silva 2003). On the one hand, WISDOM uses three models related to interaction modelling: the Interaction Model in analysis step, and Dialog Model and Presentation Model in design step. However, WISDOM does not support automatic software generation at business logic level or interface level. On the other hand, UMLi uses a User Interface Diagram based on UML to capture interaction requirements formally. UMLi project includes an automatic code generation process for user interfaces. However, the models have to be built with too much detail. Therefore, the User Interface Diagram is little practical and a medium–size specification problem is difficult to carry out.

Finally, in the web engineering field, there are several web engineering methods that, as OOWS does, use models to define web user interface. Some examples are OOHDM (Schwabe 1996), WebML (Ceri 2003) or WSDM (De Troyer 2003). However, their presentation models are mainly focused on visual appearance and configuration of web system information. As a consequence, using their presentation models in other platforms is a difficult task.

The main difference with regard to other interaction approaches mentioned above, is that the Abstract Interaction Model presented is incorporated into a software production method. Therefore, the user interface generated is a component of fully functional system.

3 The OO-Method Abstract Interaction Model

This model extends OO-Method Presentation Model with the concept called *Interaction*. In the context of this work, interaction is defined as the actions that take place between a human user and an interface, which acts as communication link to the software system functionality, in order to perform a particular task. Therefore, in the interaction process there are three main actors: the user, the software system and the interface between them. Since the software system is modelled by OO-Method Conceptual Model, the main task of the Interaction Model is to define the other two actors. As a consequence presentation model is not a precise concept to abstract interaction.

Interfaces produced by the Interaction Model follow an MDA approach. Figure 1 illustrates the global approach. First, interactions are modelled in an abstract level (PIM) using the User Model and the Abstract Interface Model (explained below). Next, the abstract models are translated to the Concrete Interaction Model (PSM) in order to capture the specific platform requirements. This paper is focused only on the abstract level. To define the two models that compose the Interaction Model abstract level, OOWS and OO-Method Presentation Models have been used as starting point. Combining experiences from both domains (Web and Desktop environments) a more expressive model has been proposed.

The main modelling constructor of the Abstract Interaction Model is the Interaction Unit (IU). An IU is defined as a modelling elements container that encapsulates a specific interaction (Buy a ticket, See all clients, etc.) between the user and the software system. Introducing this concept, the whole interaction process can be seen as an aggregation of different Interaction Units.

3.1 User Model

The User Model represents sets of human users that are allowed to interact with the system. Each type of user has the rights to access to a set of UIs which defines its Interaction Map. Therefore, a main objective of the Interaction Map is to provide a global vision about the user available interactions. To emphasize reuse, a user can inherit the Interaction Map from another User or in other words, Users can inherit IU access rights. In this case, all parent IUs are available for the child user that can additionally access to its own IUs. The notation used to define the Interaction Map is based on UML Use Cases; users are presented as actors that are connected by arrows to stereotyped UML packages that represent UIs. This notation has been chosen among others, because is widely accepted in the Software Engineering community. The figure 2 illustrates an Interaction Map.

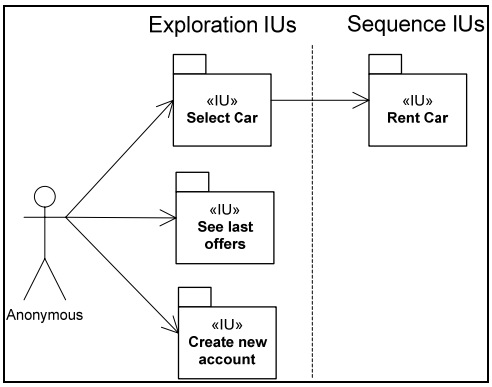

Figure 2: Interaction Map Diagram

Interaction Units are classified in the Interaction Map from an accessibility point of view: Exploration IUs that are always available to the user and Sequence IUs that only can be reached from a source IU. Sequence IUs are useful to define a sequence of previous required interactions to follow. For example, the interaction “Rent car” cannot be realized if previously, the “Select Car” interaction has not been performed.

3.2 Abstract Interface Model

This model defines for each IU the set of interaction components that define its interface with the software system. Interaction components are conceptual modelling elements that describe the interaction behaviour expected by the user but not how this interaction is implemented. For that reason, interaction components are not related to visual aspects such as colour, font size or layout, though their final implementations are user interface widgets. An IU can be composed by two types of interaction components: Basic Interaction Components (BICs), which describes a generic interaction and Interaction Patterns that models a complex interaction.

3.2.1 Basic Interaction Components

A Basic Interaction Component or BIC represents a generic interaction. The BIC concept is related to the abstract canonical components introduced in (Constantine 2003) that have been used by other approaches as USIXML (Vanderdonckt 2004). Each BIC abstracts, in a simple way, a clear purpose from the interaction point of view. Thanks to their flexibility, BICs can be used to represent quickly the interaction needed with the system. Abstract Interface Model uses five BICs:

- Input: this component models the data introduction to the system introduced by the user. Common interactions that are represented by this component are for example to write a search string, login or introduce personal data.
- Output: shows to the user information retrieved from the system. This BIC is very common in user interfaces. It models interactions such as to show information in a table, field labels or feedback messages.
- Navigation: in the Interaction Model a transition from one UI to other UI is called navigation. Therefore Navigation BIC is related to an interface element that triggers navigation. Examples in the final system are links in a web application or the main menu in a desktop one.
- Action: an action interaction component triggers an event that changes the state of the system objects or the interface. This BIC is mainly related to service execution. Push a button to store data is an interaction example that can be modelled as an action interaction.
- Group: this component groups a set of BICs in order to define a more complex component. Grouping is very useful to provide relationship between BICs. For example, in a conventional invoice service, to separate those arguments representing personal data from those arguments describing the billing information.

With the purpose of illustrate the use of BICs, a little example is explained. The IU "Create new account" (See Fig.2) is a modelled interaction to create a new customer in the System. The user must introduce the personal information (name, e-mail, password, country etc.) and choice a set of personal preferences. When the data is entered the new user is created in the system and a confirmation e-mail is sent.

This interaction is modelled with several inputs that represent the user information to be entered. The inputs can be divided into two groups: Personal Information and Profile Preferences. After that, two Action BICs are needed; one to store the information and another to send an e-mail reply. Finally a navigation component is defined in order to navigate to an IU that informs the user about the operation result.

3.2.2 Interaction Patterns

An Interaction Pattern (IP) defines a complex interaction, such as to retrieve data or to fill a form and execute a service, carried out frequently by users. The set of interactions that can be expressed by BICs are too generic so interaction behaviour must be defined at Concrete Level. For that reason, the main purpose of Interaction Patterns is to be more detailed than BICs. As a result, the analyst can define more precise interactions and more related to the domain. From a model using IPs, a functional user interface can be generated whereas from an interaction model made up by BICs, only a prototype is possible. Therefore, both concepts complement each other in order to support a wide range of user interactions.

Each IP is defined by a pattern template (Molina 2002) that describes the pattern in a generic way to be compared with others. This template is made up by five sections: 1) *Intent*: the interaction that is modelled and the problem that the patterns solve, 2) *Formal representation*: a MOF based meta-model that describes the pattern structure and its relationships with other patterns and modelling elements, 3) *Specification*: the information needed to instantiate the pattern, 4) *Semantics*: the relationship between the pattern elements and their corresponding interface and 5) *Example*: an user interface implementation that shows the pattern in action. Currently, the Abstract Interface Model is composed by ten patterns that are briefly described below:

- Population: represents a set of instances retrieved from the system that shows data to the user. The Population IP is defined as an information view over the OO-Method Class Diagram. This view is made up by a Manager Class and a set of its attributes that describes what information is retrieved. For example, if the user wants to know all names of the cars in the system, a Population IP is defined over class "Car" and its attribute "name". The information could be complemented by means of several Complementary Classes, which have a structural relationship with the Manager Class. From the Complementary Classes only the instances related to the Manager Class are shown.
- Service: abstracts a service dialog (usually a form) in order to be executed. Service IP is associated to a unique service from the Class Diagram. For each argument from the service signature, an Input BIC is created to insert the corresponding value. In addition, an Action BIC is needed to trigger the execution.
- Feedback: shows a message to the user from the system. There are three types of possible feedback behaviour: error, when an internal or validation error has happened; information, to inform the user about an specific situation; and progress, to show the evolution of a complex transaction
- Order Criteria: this pattern is always related to a Population IP. It defines how to order the in-

stances of a Population IP (Ascendant or Descendent) from a set of attributes defined in the view. This mechanism improves usability allowing the user to find information easily.

- Validation Rule: a validation rule is related to an Input BIC. It defines a rule based on a logic formula that must be accomplished by the value introduced. If the value is not correct, an error, which is defined as a Feedback IP, is shown.
- Enumeration: this pattern defines a set of values associated to an Input BIC. The user only can choose one value from the enumeration to fill the input. The set of values could be a static list of values, defined in modelling time, or a dynamic list of values linked to a Population IP.
- Filter: a filter is always related to a Population IP. By default, a Population IP retrieves all the instances that compose the view. A filter defines a well-formed formula that restricts the population to be retrieved. Only the instances that comply with the formula are shown to the user. Two types of filters are distinguished: dynamic, if the user must introduce a value to define the filter condition (as a consequence, an Input BIC is needed) or static if the condition is fully specified.
- Object Navigation: this pattern models a navigation that is triggered when an object attribute is selected within a Population IP. When the new IU is reached, the object oid is available in the target IU. This information is globally available to other IPs that made up the target IU and can be used to restrict their interactions to a particular object.
- Relationship Navigation: this navigation is associated to a relationship defined in a Population IP. When an instance of the Manager Class and the relationship is selected, the navigation is triggered. In the target IU, the object oid from the instance and the relationship id is available.
- Service Navigation: in contrast to previous navigations, this navigation is triggered when a service is executed. Therefore, service navigation is related to a Service IP. As commented before, the target IU receives the object oid from which the service was executed.

Usually, an IP extends one or more BICs or is related to another IP to complement the interaction that it offers. In addition, an IP can model behavioural aspects related to the system and the domain model (OO-Method Class Diagram). Therefore, an IP can be related to modelling elements from the OO-Method Class Diagram such as classes and its attributes, associations or operations.

4 Applying the Interaction Model: Rent a Car

In order to show the use of the Abstract Interaction Model, a small example is explained. This application example is based on an on-line rent a car service. The interactions to model are:

- Allow the user to select a car by its category from all available in the system
- Model a dialog to rent the selected car and validate the information

The user to perform the interaction is an “anonymous user”. The Interaction Map is made up of two Interaction Units, each one related to a requirement: Car Selection and Rent Car (See Fig. 2). The first IU is an Exploration IU whereas the second is a Sequence IU (a car would not be able to be rented if it had not previously selected).

The first Interaction Unit, a car list selection, is modelled by a Population IP. The view is associated to Car Class and Category Class from the Class Diagram. This Population IP includes the relevant car attributes to show to the user as: car name, rent price, description and so on. To aid the customer to select a car, a Dynamic Filter IP is defined in order to show only the cars related to the selected category. In addition, an Order Criteria IP defined over an attribute from the Population IP, for example car name, is recommended to aid the car selection attending to usability issues. Finally, an object navigation whose target is “Rent Car” IU and defined over car name attribute is specified.

Next, the second IU “Rent Car” has as main element a Service IP. This service IP is related to the operation Rent from the class Car. The Service IP is composed by several Input BICs that represent the operation arguments: car to rent, delivery date, return date, customer name, etc. The argument car to rent is filled with the car object that was previously selected in “Car Selection” IU. For a delivery and return date, two Validation Rules IP are mandatory to avoid incorrect values (for example, a previous date from today). Feedback IPs to inform user about errors or process progress are recommended according to usability guidelines.

5 Conclusions and Further Research

This poster presents a new Abstract Interaction Model for OO-Method, an MDA software generation method. This Abstract Interaction Model together with the rest of OO-Method Conceptual Models generates automatically full functional systems. To be compliant with MDA principles and HCI community, this Interaction Model is defined in two levels: 1) an abstract level that describes the interaction independently of concrete interface aspects; 2) a concrete level that defines interaction details for a concrete platform. This decision is compatible with the previous works in the field.

Two models are proposed to model interaction at abstract level: the User Model and the Abstract Interface Model. These models contribute to more expressiveness to the OO-Method Presentation Model. A small example has been used as a basic proof of concept. Currently, we are studying how a first draft of the Abstract Interaction Model, could be generated from the requirements capture phase (España 2006).

Another interesting contribution is the interaction components presented in the Abstract Interface Model: Basic Interaction Components and Interaction Patterns. Both concepts complement OO-Method to cover more interaction possibilities. On the one hand, BICs can be used to define a user interface quickly or to represent interaction that Interaction Patterns does not cover. On the other hand, the set of IPs represents complex interactions that have been extracted from real implemented applications developed with OlivaNOVA and customer requirements. Therefore, IPs could be a useful guide to solve similar problems in other model-based software development methods. As future work, the set of IPs could be extended if new interactions, which can be abstracted as patterns, are detected. To achieve this task, an empirical evaluation is planned to be done in order to detect possible lack of expressiveness.

With the purpose of producing high-quality user interface, the Interaction Model must include usability aspects defined in the ISO/IEC 9126-1 (ISO/IEC 9126, 2001). As future research, usability features must be included to guarantee that generated systems are quality systems. Once the Interaction Model has been validated with an empirical evaluation and the ISO/IEC 9126-1, the final step is to include the new Interaction Model inside the OO-Method software generation process. As a consequence, the OO-Method Model Compiler will be able to generate the user interface that implements the Abstract Interaction Model.

References

UML: OMG, UML. Unified Modeling Language, version 2.1.1. http://www.uml.org/#UML2.0. Accessed 28 Jun 2007.

MDA: OMG, MDA. Model Driven Architecture Guide. Juny 2006, http://www.omg.org/docs/omg/03-06-01.pdf. Accessed 28 Jun 2007.

Pastor, O., Gómez, J., Insfrán, E. Pelechano, V. (2001) The OO-Method Approach for Information Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming. Information Systems, 26(7) 507–534.

España, S., Panach, I., Pederiva, I., Pastor O. (2006). Towards a Holistic Conceptual Modelling-based Software Development Process. ER 2006, Arizona.pp. 437-450.

ISO/IEC 9126-1 (2001), Software engineering - Product quality - 1: Quality model.

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

Vanderdonckt, J., Q. Limbourg, et al. (2004). USIXML: a User Interface Description Language for Specifying Multimodal User Interfaces. Proceedings of W3C Workshop on Multimodal Interaction WMI'2004, Sophia Antipolis, Greece.

Silva, P.P.d. and N.W. Paton, User Interface Modeling in UMLi. IEEE Software, 2003. 20(4): p. 62-69.

Molina, P.J., Meliá, S., Pastor, O. (2002): Just-UI: A User Interface Specification Model. In: Proc. of 4th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2002. Kluwer Academic Press, Dordrecht 63–74.

CARE: Care Technologies: http://www.care-t.com Accessed 3 July 2007.

WISDOM: Nunes, N. J. y J. F. e. Cunha (2000). "Wisdom: a software engineering method for small software development companies." Software, IEEE 17(5): 113-119.

UMLi: da Silva, P. P. d. and N. W. Paton (2003). "User Interface Modelling in UMLi " IEEE Softw. 20 (4). pp. 62-69.

OOHDM: Schwabe D., Rossi G., and Barbosa. S. (1996) Systematic Hypermedia Design with OOHDM. In ACM Conference on Hypertext, Washington, USA.

WebML: Ceri, S. Fraternali, P., Bongio, et al. (2003). Designing Data-Intensive Web Applications.Morgan Kaufman.

WSDM: De Troyer, O. and Casteleyn, S. (2003) Modelling Complex Processes from web applications using WSDM. In IWWOST 2003. Oviedo, Spain. pp 1-12.

OOWS: Fons J., P. V., Albert M., and Pastor O. (2003). Development of Web Applications from Web Enhanced Conceptual Schemas. ER 2003, LNCS. Springer.pp. 232-245.

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

Giner, P., V. Torres, and V. Pelechano. Building Ubiquitous Business Process following an MDD Approach. in XII Jornadas de Ingeniería del Software y Bases de Datos . 2007 (Pending Publication). Zaragoza, Spain.

Valverde, F., et al. A MDA-Based Environment for Web Applications Development: From Conceptual Models to Code. in 6th International Workshop on Web-Oriented Software Technologies (Pending Publication). 2007. Como (Italy).