AUTOMATIC GENERATION OF USER INTERFACE MODELS AND PROTOTYPES FROM DOMAIN AND USE CASE MODELS

António Miguel Rosado da Cruz
E.S.T.G., Instituto Politécnico de Viana do Castelo, Av. do Atlântico, s/n, 4900-348 Viana do Castelo, Portugal

João Pascoal Faria
Faculdade de Engenharia da Universidade do Porto/INESC Porto, Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal

Keywords: Model-driven software development, Use case model, Domain model, Automatic user interface generation, Iterative incremental process, Form-based business applications.

Abstract: The model-driven automatic generation of interactive applications has been addressed by some research projects, but only few propose the model-to-model generation of a graphical user interface (UI). Existing solutions generate only part of the interactive application and most of them require as input the full specification of a UI model. This paper proposes an iterative and incremental approach that enables the modeler to generate a form-based executable prototype from the constructed models, favouring an evolutionary construction of models starting with a domain model, proceeding with an extended domain model and finally complementing it with a use case model. The approach derives a UI model from the previously referred models and allows its execution by generating an executable description of the UI in a XML-based UI description language, together with code for the specified logic and for persisting the data entities. The generated UI description may be further refined and supplemented with style definitions in order to obtain a final UI.

1 INTRODUCTION

Model-driven software development (MDD) approaches, like Domain Specific Modeling – DSM – (Kelly and Tolvanen, 2008), or the OMG's Model Driven Architecture – MDA – (Warmer et al., 2003), are based on the successive refinement of models and on the automatic generation of code and other sub-models. This paper presents an approach for the automatic generation of form-based applications within a model-driven software development setting. The approach proposed involves the iterative and incremental development of a domain model, and optionally a use case model, by the modeler, and the testing of an automatically generated executable prototype.

In order to disambiguate and raise the rigour of the models, the domain model is enriched with OCL constraints, from which the generation process takes advantage to produce validation procedures in the user interface (UI).

In the next sections, the proposed approach is presented focusing on the features that are derived in the generated interactive application or its UI, and the model characteristics that are explored in order to derive those features. The relations between the three metamodels involved in the process are analysed, namely an extended domain metamodel, a use case metamodel and a user interface metamodel. The extended domain metamodel extends the domain metamodel with derived attributes, derived classes (views) and user defined operations and triggers. An example will help to perceive the differences between a domain-model only approach to modeling, developed during simple domain analysis and addressed in (Cruz and Faria, 2008), and a use case driven approach followed when eliciting and modeling requirements, within a Unified Process-like software development process (Jacobson et al., 1999). In the latter case the use case model must be constructed in close connection with the extended domain model, referring to its classes (base or derived) and operations (user-defined or

Rosado da Cruz A. and Pascoal Faria J. (2009).
AUTOMATIC GENERATION OF USER INTERFACE MODELS AND PROTOTYPES FROM DOMAIN AND USE CASE MODELS.
In *Proceedings of the 4th International Conference on Software and Data Technologies*, pages 169-176
DOI: 10.5220/0002253501690176
Copyright © SciTePress

pre-defined CRUD operations – create/retrieve/ update/delete). An example is presented in section 6.

Before concluding the paper, related work is addressed and compared to the presented approach.

2 GENERAL APPROACH

The goal of our approach, illustrated in Figure 1, is to allow the automatic generation of user interface models (UIM) and executable user interface prototypes (UIP), from early, progressively enriched, system models.

In the first iterations, a simple domain model (DM) is constructed, represented by a UML class diagram, with classes (domain entities), attributes and relationships. From this DM a simple UI can be automatically generated (by the EDM2UIM process, a model to model transformation, and M2C, model to code transformation, in Figure 1) supporting only the basic CRUD operations and navigation along the associations defined.

In subsequent iterations, the domain model is extended with additional features (to be explained in more detail in section 4) that allow the generation of richer user interfaces: OCL constraints, default values, derived attributes, derived classes (views), user-defined operations, and triggers. Indeed, lists of possible field values can be generated from OCL class invariants, and operations' pre-conditions will influence what the user is able to do in the generated user interface. Derived classes allow the generation of UI forms with a more flexible data structure.

Simultaneously, the modeler may develop a use case model (UCM), integrated with the extended domain model (EDM). This UCM will enable the separation of functionality by actor, and its customization (e.g.: hiding functionality for some actors). Corresponding UI models and prototypes are then automatically generated from both the EDM and UCM (EDM+UCM2UIM and M2C processes in Figure 1). As will be explained in section 5, there is a full integration between the UCM and EDM, as use case specifications are established over the structural domain model.

On each iteration, the generated UI may be tuned by a UI designer in two points of the process: after having generated an abstract UIM, but before generating a concrete UI; and, after generating a concrete UI in a XML-based UI description language (e.g.: XUL), which allows for the a posteriori customization and application of style sheets. A proof of concept tool has been developed for fully automating the EDM2UIM, EDM+UCM2UIM and M2C processes. The prototyped M2C process uses XUL to represent a concrete executable UI description, JavaScript for the executable functionality and RDF to persist data.

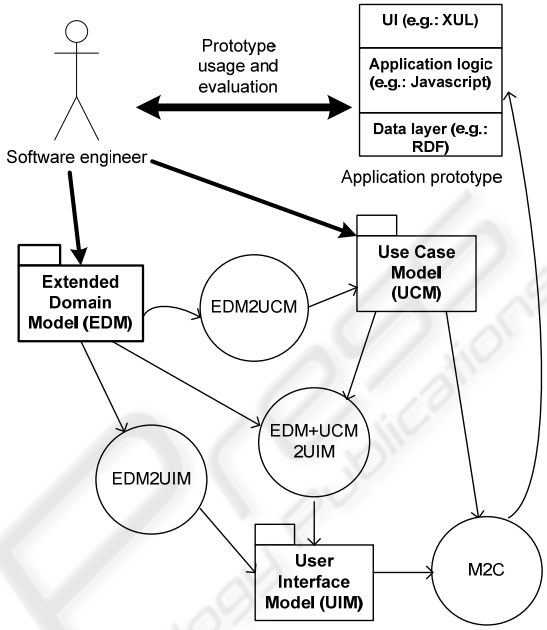

Figure 1: General approach to UI generation.

3 CONCEPTUAL META-MODEL

Each of the models (EDM, UCM and UIM) presented in Figure 1 is an instance of a defined metamodel, of which an excerpt is shown in Figure 2 (EDMM, UCMM and UIMM, respectively). Elements in the user interface model are traced back to elements in the UCM or EDM, e.g.:

- A Menu in the UI traces back to a Use Case (UC) Package in the UCM;
- a Menu Item traces back to a top-level UC in the UCM, i.e. a UC that directly links to an actor;
- A Form can be traced back to a UC, which is always related to a base or derived domain Entity;
- An Action Button may trace back to a CRUD operation that may be identified in a UC, or to a UC that extends another UC and has an associated user defined operation.

In the next section the mappings for deriving a UI model from one or both the other models (EDM and UCM), as depicted in figure 1, are defined.

A set of rules has also been defined for transforming an EDM into a default UCM (EDM2UCM process), but these are not presented

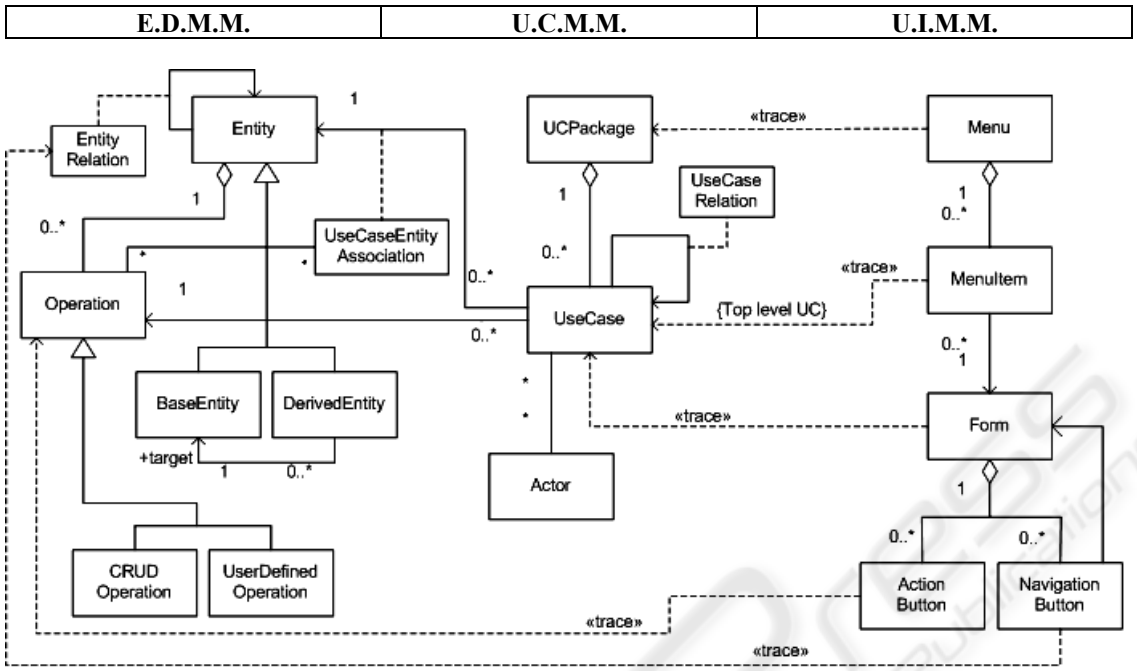

Figure 2: Excerpt of the conceptual metamodels and their relations.

due to space reasons. The default use case model has only one actor that has access to all the system functionality, and may serve as the basis for producing the intended use case model, by eliminating or redistributing functions among actors.

4 EDM FEATURES AND TRANSLATION TO THE UIM

Besides classes (domain entities), attributes and relationships, an **extended domain model** may contain the following features:

- **Class Invariants.** intra-object (over attributes of a single instance) or inter-object (over attributes of multiple instances of the same or related classes) constraints defined in a subset of OCL.
- **User-defined Operations.** Operations defined in an Action Semantics-based action language, supplementing the basic CRUD operations (Create, Retrieve, Update and Delete).
- **Derived Attributes.** Attributes whose values are defined by expressions in a subset of OCL, over attributes of self or related instances. A common special case is a reference to a related attribute, using a sequence of dot separated names.
- **Default Values.** Initial attribute values defined in a subset of OCL.
- **Derived Classes (views).** Classes that extend the domain model with non-persistent domain entities with a structure closer to the UI needs. Currently, each derived class must be related to a target base class, and is treated essentially as a virtual specialization of the base class, possibly restricted by a membership constraint and extended with derived attributes.
- **Triggers.** Actions to be executed before, after or instead of CRUD operations, or when a condition holds within the context of an instance of a class. By defining triggers, the modeler is able to modify the normal behavior of CRUD operations, or define generic business rules.

The main transformation rules for generating a user interface model from an extended domain model are summarized in table 1. Rules for transforming simple domain models were previously addressed in (Cruz and Faria, 2008).

When the UIM/UIP is generated solely from the domain model, a special class named *System* has to be created and linked to the domain classes that should correspond to the application entry points. A more flexible approach is explained in the next section.

Table 1: EDM to UIM/UIP transformation rules.

EDM feature	Generated UI feature (UIM/UIP)
Domain entity	Form with an input/output field for each attribute, and buttons and associated logic for the CRUD operations.
Inheritance	A field for each inherited attribute in the form generated for the specialized class.
To-many association, aggregation or composition	UI component in the source class form, with a list of the identifying attributes of the related instances of the target class, and buttons for adding new instances and for editing or removing the currently selected instance.
To-one association, aggregation or composition	Group box in the source class form, with a field for each identifying attribute of the related instance. If the related instance is not fixed by the navigation path followed so forth, then a button is also generated for selecting the related instance.
Enumerated type	Group of radio buttons for selecting one option.
Class invariant	Validation rule that is called when creating or updating instances of the class.
User-defined operation	Button and associated logic, within the form corresponding to the class where the operation is defined. Forms are also generated for entering the input parameters and displaying the result, in case they exist. The operation pre-condition determines when the button is enabled.
Derived attribute	Output-only field (calculated field).
Default value	Initial field value.
Derived class (view)	Form with an input/output field for each attribute of the target class, an output-only field for each derived attribute, and buttons for the CRUD logic (over the target class).
Operation-Action Trigger	Logic that is executed before, after or instead of the CRUD operation that it refers to.
Condition-Action Trigger	Logic that is executed every time the condition holds, after creating or updating an instance of the class where the trigger is defined.

5 UCM FEATURES AND TRANSLATION TO THE UIM

In our approach, a UCM can be defined in close connection with the EDM, to indicate and organize the CRUD, user-defined or navigational operations over base or derived domain entities that are available for each actor (user role). The data manipulated in each use case is determined by the domain entity and/or operation associated with it. Several constraints are posed on the types of use cases and use case relationships that can be defined.

Two categories of use cases are distinguished:

- **Independent use Cases:** use cases that can be initiated directly, and so can be linked directly to actors (that initiate them) and appear as application entry points;
- **Dependent use Cases:** use cases that can only be initiated from within other use cases, called *source* use cases, because they depend on the context set by the source use cases; the dependent use cases extend or are included by the source ones, according to their nature (optional or mandatory, respectively).

The types of independent use cases that can be defined in connection with the EDM are:

- **List *Entity*.** view the list of instances of an entity (usually only some attributes, marked as identifying attributes, are shown);
- **Create *Entity*.** create a new instance of an entity;
- **Call *StaticOperation*.** invoke a static user-defined operation defined in some entity; this includes entering the input parameters and viewing the results, when they exist.

The types of dependent use cases that can be defined in connection with the EDM are:

- **Retrieve, Update and/or Delete *Entity*.** view (retrieve) or edit (update or delete) an instance of the entity previously selected (in the source use case);
- **Call *InstanceOperation*.** invoke a user-defined operation over an instance of an entity previously selected (in the source use case); this includes entering the input parameters and viewing the results, when they exist;
- **List Related *Entity*.** view the list of (0 or more) instances of the target entity that are linked to a previously selected source object (in the source use case); in case of ambiguity, in this and in the next use case types, the link type (association) must also be specified;
- **Create Related *Entity*.** create a new instance of the target entity type and link it to a source object previously selected (in the direct or indirect source use case);

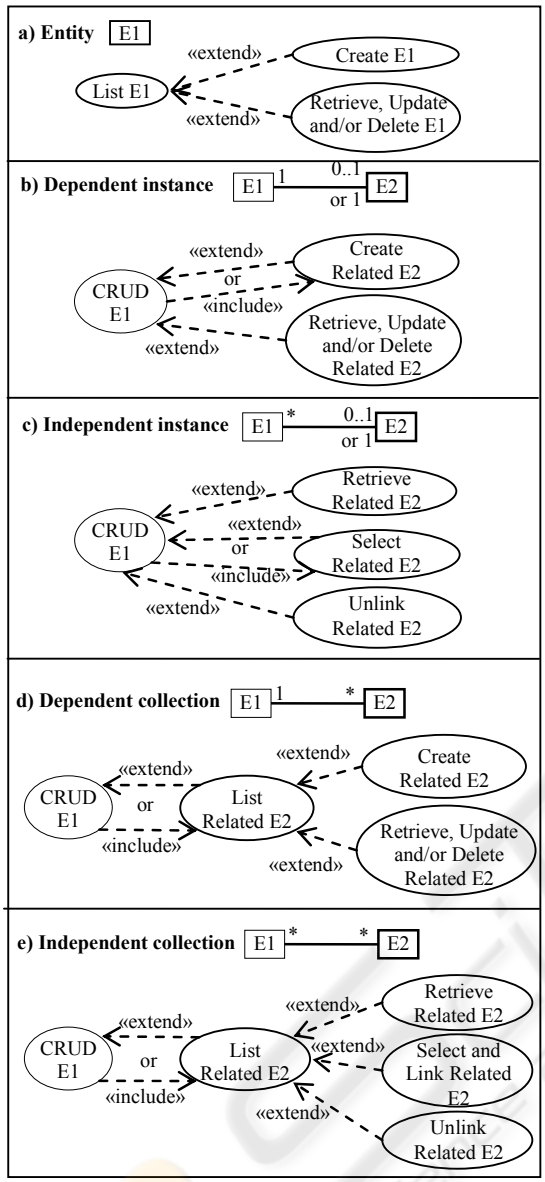

Figure 3: Possible types of relationships among use cases for different domain model fragments (note: aggregations and compositions are treated similarly to associations).

- **Retrieve, Update and/or Delete Related *Entity*.** view (retrieve) or edit (update or delete and unlink) the instance of the target entity type that is linked with a source object previously selected (in the direct or indirect source use case);
- **Select Related *Entity*.** select (and return to the source use case) an instance of the target entity that can be linked to a source object previously selected (in the source use case);
- **Select and Link Related *Entity*.** select an instance of the target entity and link it to the source object previously selected (in the source use case);
- **Unlink Related *Entity*.** unlink the currently selected instance of the target entity (in the source use case) from the currently selected source object (in the source use case).

Table 2: UCM to UIM transformation rules.

UCM feature	Generated UI feature (UIM/UIP)
Actor	Button in the application start window, linking to the actor's main window.
Use Case Package	Menu in the actor's main window, with a menu item for each use case that belongs to the package and is directly linked to the actor.
Use Case of type *List Entity* or *List Related Entity*	Form that displays the full list of instances or the list of related instances of the target entity, with buttons for the allowed operations (according to the dependent use cases). Only the identifying attributes are shown.
Use Case of type *Select Related Entity* or *Select and Link Related Entity*	Form that displays the list of candidate instances and allows selecting one instance. Only the identifying attributes are shown.
Use Case of type *CRUD Entity* or *CRUD Related Entity*	Form that displays the object attribute values, with buttons and functionality corresponding to the CRUD operations allowed. In the case of a related instance, the identifying attributes of the source object are shown but cannot be edited.
Use Case of type *Call User-Defined Operation*	Forms for entering and submitting input parameters and presenting output parameters, when they exist.
Extend relationship	Button in the form corresponding the base use case that gives access to the extension.
Include relationship	If the included use case is of type "*List...*", it is generated a sub-window. Otherwise, it is generated a button in the source use case.

The entity, operation(s), and link type (when needed) associated to each use case are specified with tagged-values.

The types of relationships that can be defined among use cases are illustrated in Figure 3.

Table 2 summarizes the rules for generating UI elements from the UCM. Their application is illustrated in the next section.

6 EXAMPLE

This section presents an example of a Library System that illustrates the approach. Figure 4 shows the constructed EDM. Such model has been developed in several iterations; an executable prototype has been automatically generated and tested at the end of each iteration. After having a partial or complete EDM, the modeler may also develop a UCM. Figure 5 illustrates an extract of a UCM that was developed for this system. Table 3 shows the entity types and operations associated (via tagged values) with some of the use cases. By applying the mapping rules described in the previous sections, the EDM+UCM2UIM process generates a UI model and then an executable prototype, part of which is shown in Figure 6.

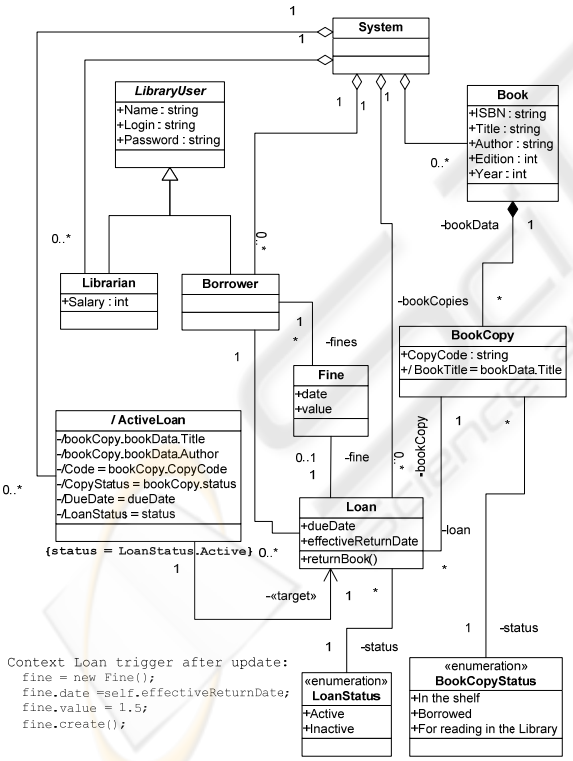

Figure 4: Extended domain model (EDM) for a Library Management System, with an example trigger.

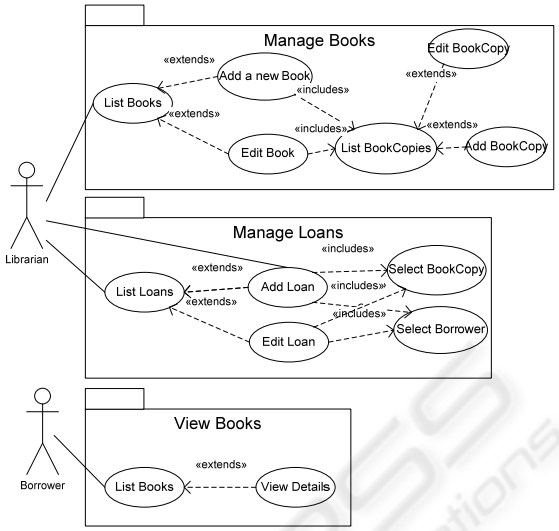

Figure 5: Partial use case model (UCM) for the Library Management System.

Table 3: Entities and operations associated (via UML tagged values) with some of the use cases in Figure 5.

Use case	Entity	Operation(s)
List Books	Book	List
Add a new Book	Book	Create
Edit Book	Book	Update
List BookCopies	BookCopy	List Related
Add BookCopy	BookCopy	Create Related
Edit BookCopy	BookCopy	Update Related, Delete Related
Select BookCopy	BookCopy	Select Related
View Details	Book	Retrieve

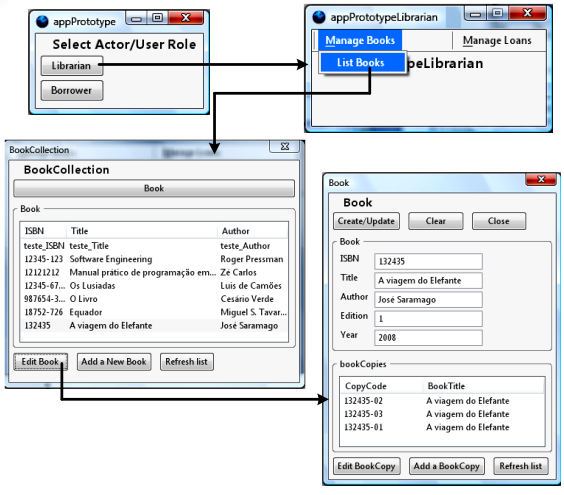

Figure 6: Excerpt of the application prototype generated for a Librarian executing use cases List Books → Edit Book (that includes List BookCopies).

7 RELATED WORK

Few approaches found in the literature allow a model-to-model generation of a graphical user interface, within a MDD setting. The XIS profile and method (Silva, 2003; Silva et al., 2007; Silva and Videira, 2008), just like the OO-Method/Olivanova (Pastor and Molina, 2007) and the ZOOM approach (Jia et al., 2005, Jia et al., 2007) are able to produce a fully functional (executable) application, but the demanded input models are very time consuming and arduous to build.

The need to attach a stereotype to every model element, in XIS, makes the models hard to read and build. Unlike XIS, our approach doesn't demand the stereotyping of every model element, as the full model package is submitted to the transformation process.

XIS allows two approaches to interactive systems generation: a dummy approach, where a domain model, an actors' model, and a UI model must be fully specified; and the XIS smart approach, that enables the derivation of the UIM, called the user interfaces view, by demanding the construction of two other models, a business entities model and a use case model. This approach to the UIM derivation is simpler than its full construction, but it comes with the cost of the inflexibility of the generated UI.

XIS business entities select domain entities relations to provide a lookup or master/detail pattern to the UI needed for the interaction inside the context of a use case (Silva, 2003; Silva et al., 2007).

XIS business entities are similar to our derived entities. Like in the XIS smart approach, the modeler must attach to each use case an Entity (base or derived) from the EDM. The difference is that, in our approach, relations between entities are inferred from the EDM, thus not being needed a separate business entities model to provide higher level entities to the UCM. The relation's selection provided by the XIS business entities model can be done, within our approach, in the UCM by not modeling UC for navigating through the relations.

Similarly to XIS and the OO-Method, in our approach CRUD operations are predefined.

It is not possible, in XIS, to specify complex behavior - only CRUD operations may be used attaching it to Business Entities and to the connection between the UCs and business entities.

In the OO-Method user defined operations (services and transactions) can be specified by using OASIS (Pastor and Molina, 2007). Also, for not demanding the knowledge of OASIS, the OO-Method has a solution that comes with the cost of inflexibility: it is possible to specify how each service changes the object state depending on the arguments of the involved service and the current object state, by categorizing every attribute in one of three categories, and introduce the relevant information depending on the corresponding selected category. Possible attribute categories are push-pop, state-independent, and discrete-domain valued attributes. The OO-Method permits, as well, the specification of allowed states and state transitions within a class. Each state transition may have attached a control (guard) or triggering condition. It is also possible to define transactions involving services of different classes.

In our approach user defined operations may be specified using an UML Action Semantics-based language.

Just like our approach, the OO-Method allows the definition of derived attributes, by assigning a calculation formula to the attributes.

The ZOOM approach models a system using the ZOOM language, which is a formal object oriented extension of Z. Additionally it allows the building of a graphical model, which is then translated to ZOOM. The models that are demanded by the approach, in order to automatically generate an executable application, are (Jia et al., 2005): a class model, which models the structure of the system and contains all the classes of the application; a finite state machine model that models the system behavior and is the central communication mechanism to connect the structural model to the UI model; and, a UI model, which models the UI screens by using predefined components that are organized according to a user defined layout.

Elkoutbi et al. (Elkoutbi et al., 2006) and Martinez et al. (Martinez et al., 2002) approaches are able to produce a UI from the structural, use case and UI behavioral models, but demand the attachment of UI related information (input/output fields and/or widgets) to collaboration diagrams and message sequence charts used to specify use case behavior. The generated output is only able to simulate the specified use cases through the generated UI, with no business level application behavior.

Forbrig et al. approach (Forbrig et al., 2004) and Wisdom (Nunes, 2001) are not automatic. Forbrig et al. base their approach on the manual selection of patterns, from a repository, that drives the model construction and transformation towards a final application. In Wisdom, the Winsketch tool

(http://apus.uma.pt/~winsketch) helps building and validating Wisdom models, and supports the tracing of model elements through the different process phases. Despite this, no code generation is done.

8 CONCLUSIONS AND FUTURE WORK

The presented approach enables a gradual approximation to system modeling, by being able to derive a default UI and an executable prototype from the DM alone, an EDM or from the EDM and the UCM. It is also possible to have these initial models in different levels of abstraction or rigour, and refine them in an incremental and iterative manner.

As depicted in section 2, this approach is able to generate a UI model from the system's non-UI submodels, helping the modeler in creating a system model for the final application. The approach derives a default UI and an executable prototype from the domain model alone, turning possible to interactively evaluate the system model with the end users, and to iteratively evaluate and refine the model. It also allows to add rigour and model elements to the system model, generating refined UIs and refined executable prototypes that support an evolutionary model-driven development with the close participation of the end users.

The main contributions of our work are to take advantage of class invariants and operation pre-conditions to generate validation routines in the executable application, enabling the enhancement of the usability of the generated UI by helping the user in entering valid data into forms, and by giving feedback identifying invalid data; and, the use of an action language to specify the semantic of operations at class level, and enable the definition of triggers activated either by the invocation of a CRUD operation or by the holding of a given state condition.

As future work, we intend to refine the definition of complex UCs with pre-/post-conditions, that will enable workflow definitions. Also the validation of the approach will be further accomplished by using more representative case studies.

ACKNOWLEDGEMENTS

We would like to thank Rui Gomes, Hugo Sereno, Filipe Correia and André Restivo for their comments and suggestions on the first versions of this article.

REFERENCES

Cruz, A. M. R., Faria, J. P., 2008. Automatic generation of interactive prototypes for domain model validation. In Proceedings of the Int'l Conference on Software Engineering and Data Technologies (ICSoft 2008), vol. SE/GSDCA/MUSE, pp 206-213. INSTICC Press.

Elkoutbi, M., Khriss, I., Keller, R. K., 2006. Automated prototyping of user interfaces based on UML scenarios. Journal of Automated Software Engineering, 13(1):5-40, January 2006.

Forbrig, P., Dittmar, A., Reichart, D., Sinnig, D., 2004. From models to interactive systems tool support and XIML. In Proceedings of the First Int'l Workshop MBUI 2004, vol. 103-CEUR Workshop Proceedings, Funchal, Portugal. Available at http://ceur-ws.org.

Jacobson, I., Booch, G., Rumbaugh, J., 1999. The Unified Software Development Process. Addison-Wesley.

Jia, X., Steele, A., Liu, H., Qin, L., Jones, C., 2005. Using ZOOM approach to support MDD. In Proceedings of the 2005 Int'l Conference on Software Engineering Research and Practice (SERP'05), Las Vegas, USA.

Jia, X., Steele, A., Qin, L., Liu, H., Jones, C., 2007. Executable visual software modelling - the ZOOM approach. Software Quality Control, 15(1):27-51.

Kelly, S., Tolvanen, Juha-Pekka, 2008. Domain Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Press.

Martinez, A., Estrada, H., Sánchez, J., Pastor, O., 2002. From early requirements to user interface prototyping: A methodological approach. In Proceedings of the 17th IEEE Int'l Conf. on A.S.E., pp 257-260.

Molina, P. J., Hernández, J., 2003. Just-UI: Using patterns as concepts for IU specification and code generation. In Perspectives on HCI Patterns: Concepts and Tools (CHI'2003 Workshop).

Molina, P. J., 2004. User interface generation with olivanova model execution system. In IUI '04: Proceedings of the 9th Int'l Conference on Intelligent User Interfaces, pages 358-359, NY, USA. ACM.

Nunes, N. J., 2001. Object Modeling for User-Centered Development and User Interface Design: The Wisdom Approach. PhD thesis, University of Madeira, July.

Pastor, O., Molina, J., 2007. Model-driven Architecture in Practice – A software production environment based on Conceptual Modeling. Springer-Verlag.

Silva, A., 2003. The XIS approach and principles. In IEEE Computer Society, Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO '03).

Silva, A., Videira, C., 2008. UML, Metodologias e Ferramentas CASE, vol. 2. Centro Atlântico, 2nd ed.

Silva, A. R., Saraiva, J., Silva, R., Martins, C., 2007. XIS - UML profile for extreme modeling interactive systems. In 4th Int'l Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES 2007). IEEE, March.

Warmer, J., Bast, W., Pinkley, D., Herrera, M., Kleppe, A., 2003. MDA Explained – The Model Driven Architecture: Practice and Promise. Addison-Wesley.