A Metamodel-Based Approach for Automatic User Interface Generation

Conference Paper · October 2010

DOI: 10.1007/978-3-642-16145-2_18

2 authors:

António Miguel Rosado da Cruz
Instituto Politécnico de Viana do Castelo
62 PUBLICATIONS **685** CITATIONS
SEE PROFILE

João Pascoal Faria
University of Porto
74 PUBLICATIONS **763** CITATIONS
SEE PROFILE

A Metamodel-based Approach For Automatic User Interface Generation

António Miguel Rosado da Cruz[1] and João Pascoal Faria[2]

[1] ESTG-Instituto Politécnico de Viana do Castelo, Av. do Atlântico, s/n 4900-348 Viana do Castelo, Portugal,
miguel.cruz@estg.ipvc.pt
[2] Faculdade de Engenharia da Universidade do Porto / INESC Porto, Rua Dr. Roberto Frias, s/n 4200-465 Porto, Portugal,
jpf@fe.up.pt

Abstract. One of the advantages of following a MDA-based approach in the development of interactive applications is the possibility of generating multiple platform-specific user interfaces (UI) from the same platform independent UI model. However, the effort required to create the UI model may be significant. In the case of data-intensive applications, a large part of the UI structure and functionality is closely related with the structure and functionality of the domain entities described in the domain model, and the access rules specified in the use case model. This paper presents an approach to reduce the effort required to create platform independent UI models for data intensive applications, by automatically generating an initial UI model from domain and use case models. For that purpose, UML-aligned metamodels for domain and use case models are defined, together with a MOF-based metamodel for user interface models. The transformation rules that drive the UI model generation are introduced. It is also proposed a MDA-based process for the development of data intensive interactive applications based on the proposed model architecture and transformations.

Keywords: MDD, MDA, Metamodel, User Interface Automatic Generation, Model Transformation.

1 Introduction

Model-driven development (MDD) is mainly focused on platform independent modeling activities rather than programming activities. This allows software engineers to focus on concepts of the problem domain, and the way they shall be modeled in order to produce a software solution, rather than being distracted by technical issues of the solution domain. Within an MDD setting, code can be automatically generated from models to a great extension, dramatically reducing the most costly and error-prone aspects of software development [1].

Model-driven development of interactive applications enables the generation of multiple platform-specific user interfaces (UI) from the same platform independent UI model. However, the effort required to create the UI model may be significant. In

the case of data-intensive applications, a large part of the UI structure and functionality is closely related with the structure and functionality of the domain entities described in the domain model, and the access rules specified in the use case model. This paper presents an approach to reduce the effort required to create platform independent UI models for data intensive applications, by automatically generating an initial UI model from domain and use case models.

The approach presented is based on a well identified subset of UML that permits the construction of a complete and rigorous platform independent model (PIM) of a software system (including constraints and actions), and extends that UML subset with new metamodel elements, that turn possible the model-driven automatic generation, by model transformation, of a User Interface Model (UIM), and the model-to-code generation of a final application from the rigorous PIM. The generation of a PIM level UIM allows its configuration and/or modification prior to generating the final code. The main contributions of this paper are:

- the definition of a process for the automatic generation of user interface models and executable prototypes from domain and use case models;
- the definition of a UI metamodel that allows the platform independent modeling of the UI structure and of the bindings from the UI structure to the domain model, therefore providing a set of models that contains all the information for generating a fully executable prototype;
- extensions to the UML metamodel, that better enable taking profit of the model features when generating the UIM; and,
- the description of a set of transformation rules that allow the derivation of a default UIM from the Domain Model (DM) and Use Case Model (UCM).

In previous work [2, 3], we informally defined extensions to the DM and UCM to better support UI generation, and explained how DM and UCM features could be mapped into UI features. In this paper, we formalize the extensions to the DM and UCM by specifying the metamodels that they shall conform to, formalize the UIM metamodel, and define the transformation rules in terms of those meta-models.

Although not detailed in this paper, the research work that yielded the presented approach is supported by a proof of concept tool and has been validated through two case studies [4].

The proposed process is presented in the next section together with the contextualization of the presented metamodels within the UML metamodel. In the following sections, the metamodels for DM, UCM, and UIM are introduced. Section 6 presents the transformation rules for automatically obtaining a UIM from the DM and UCM. Section 7 briefly overviews related work. Finally, some conclusions are drawn together with some proposals for future work. A running example is used along the paper to illustrate the approach.

2 Proposed Generation Process and Model Architecture

The approach proposed in this paper comprises an iterative development process that enables the automatic generation of UI models from early, progressively enriched, platform independent system models, and a metamodel defining the

concepts that allow a platform independent system modeling according to 3 views: a structural view, that is established through a DM; a functional view, defined by a UCM; and, a user interface view, defined through a UIM. After the modeling activity and the UIM generation step, in each iteration, the approach permits the generation of an executable user interface prototype (UIP), which enables the complete model validation by other stakeholders besides the modeler himself.

The DM represents the business entities and events of the problem domain, just like intended by the Unified Process [5].

The process, illustrated in Fig. 1, starts with the construction of a UCM and a DM that conform to a metamodel that specializes and extends UML [6, 7]. A simple UI can be automatically generated from the DM specification (by a model to model transformation process from DM to UIM, followed by a model to code transformation) supporting only the basic CRUD operations (Create, Retrieve, Update and Delete) and navigation along the associations defined. In subsequent iterations, the DM can be refined, and more information can be added to it, allowing the generation of richer user interfaces, and application prototypes.

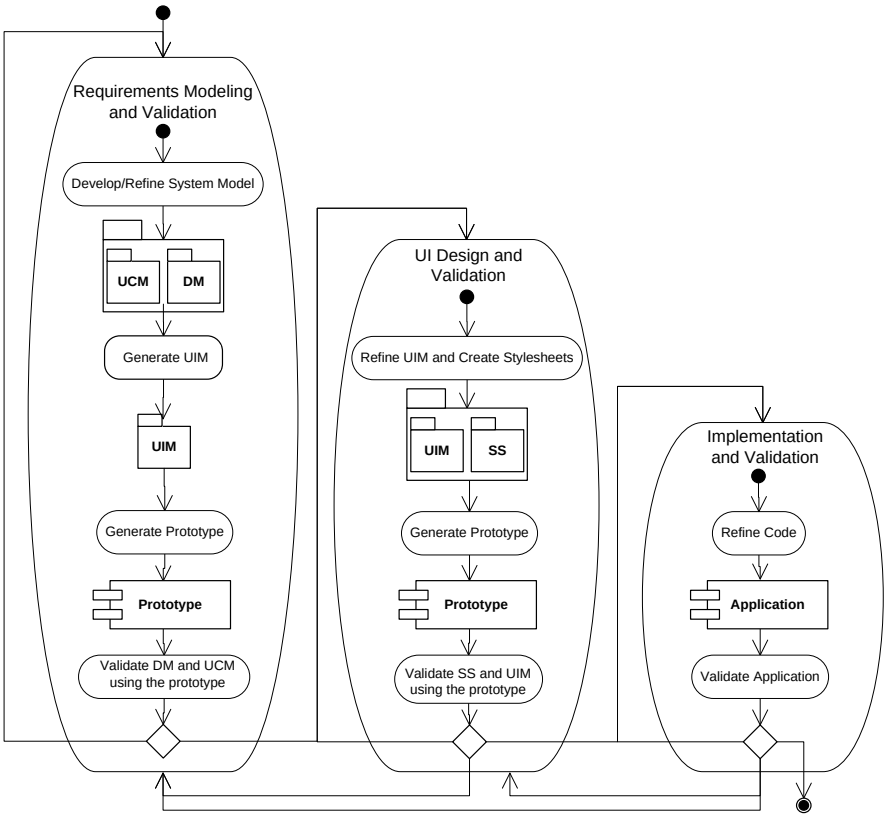

Fig. 1. Proposed generative UI development process.

When desired, the modeler may develop a UCM. In the proposed approach, there is a full integration between the UCM and the DM, as use case specifications are established over the structural DM. The UCM enables the separation of functionality by actor, and its customization (e.g.: hiding functionality for some actors). A corresponding UIM and a UIP are then automatically generated from both the DM and the UCM.

Figure 2 contextualizes the metamodel defined for this approach by dividing it in three packages corresponding to the referred model views and relating them to the UML and MOF.

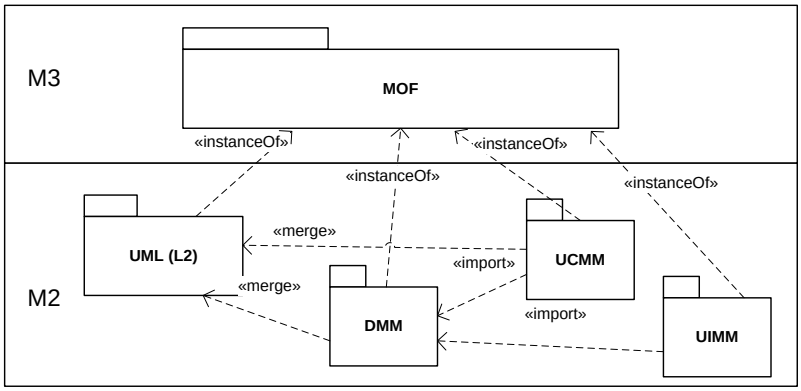

Fig. 2. Metamodel contextualization.

MOF provides the concepts for defining the UML metamodel, which is stratified in language units used for defining compliance levels[6, 7]. Compliance level 0 (L0) has a single language unit that enables modeling of class-based structures, and is formally described as the UML Infrastructure, or package Core, which is shared between MOF and the UML. Level 1 (L1) adds language units and extends the capabilities of L0 for use cases, interactions, structures, activities and actions, and L2 adds language units and extends the capabilities of L1 for state machine modeling and profiles.

The Domain Metamodel (DMM) and the Use Case Metamodel (UCMM) are partially merged with UML L2, incrementally adding features to some of its elements, by specializing or redefining other metamodel elements [7].

The User Interface Metamodel (UIMM) is defined conforming to the MOF meta-metamodel and imports features from the DMM, which enable model integration.

3 Metamodel for Domain Models

The metamodel for domain models is depicted in Figures 3 and 4, and specializes and extends the UML language unit for class diagrams [6]. The reused UML elements are shaded and the ones that have been modified, either by adding or specializing features, have only the name compartment shaded. The modified UML elements are Class (with an alias “Entity”), Property and Operation. Entity (Class) has a new

attribute, isNavigationRoot, that enables the identification of an entry point for navigating in the structure. This is useful when generating a UIM from the DM alone, without the development of a UCM. BaseEntity and DerivedEntitiy specialize the modified UML Class and inherit all its features, semantics and concrete notation. The BaseEntity models a problem domain persistable concept in a platform independent manner. Property has a new attribute, isIdent, which enables the identification of properties that are used by the business user as an instance identification or summarization structural feature. This is different from a unique identifier.

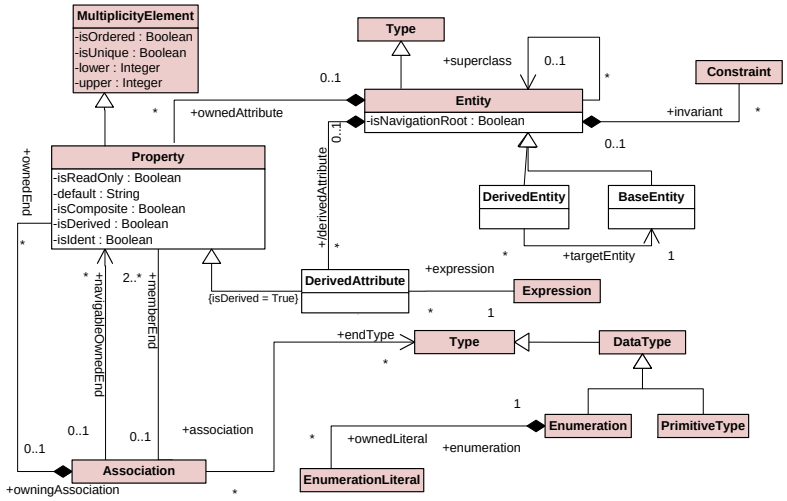

Fig. 3. Metamodel for Domain Models (structural features).

Operation is extended in order to comprise, besides operations that are user-defined in an Action Semantics-based language, the basic CRUD operations (CRUDop) that are considered to exist by default for instances of every BaseEntity (see Fig. 4).
The DM metamodel adds new model elements to the UML metamodel, easing the purpose of constructing a complete and rigorous PIM. DerivedEntity and DomainTrigger deserve further explanation.

3.1 Derived Entities

A DerivedEntity models an interesting view in the problem domain (e.g. a business view). DerivedEntities are non-persistent domain entities with a structure closer to the business domain, like a business document, and so closer to the UI needs. A DerivedEntity must target a BaseEntity that acts as the root for referencing derived attributes. It is treated essentially as a virtual specialization of the target BaseEntity, possibly restricted by a membership constraint and extended with derived attributes. It may also hide attributes from the target BaseEntity. A Derived Entity may be distinguished by having its name preceded by a slash in a concrete notation.

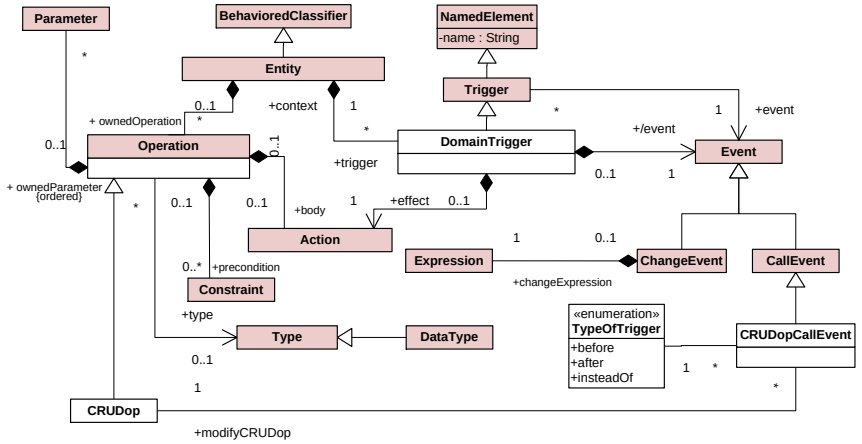

Fig. 4. Metamodel for Domain Models (behavioral features).

3.2 Domain Triggers

The DM metamodel includes constructs for defining domain triggers, which are used to capture generic business rules or to modify the default behavior of CRUD operations (CRUDop). Domain triggers are defined in the context of an Entity (see Fig. 4). A DomainTrigger inherits from the UML Trigger, but has only two possible kinds of associated events:

- **ChangeEvent:** It is the UML ChangeEvent class but with the changeExpression association end restricted to Expression type. In standard UML it can be a ValueSpecification, that includes the possibility of defining an OpaqueExpression, which promotes the definition of platform specific expressions (e.g. Java expressions) within a PIM, which is considered to be a bad modeling practice [1]. The ChangeEvent triggers a DomainTrigger when the condition defined in the changeExpression holds.
- **CRUDopCallEvent**. It is a specialization of UML's CallEvent, restricted to CRUD operations, and with the possibility to intercept the call to an operation before, after or instead of calling it. It provides a way of modifying the default behavior of CRUD operations. A CRUDopCallEvent triggers a DomainTrigger before, after or instead of an identified CRUD operation call, within the context of an instance of a class, enabling the reinforcement of business rules.

Fig. 5 shows the DM constructed for an example Library System that will help illustrate the approach along the paper. The DM has been developed in several iterations and, as defined in the process presented in section 2, an executable prototype has been automatically generated and tested at the end of each iteration.

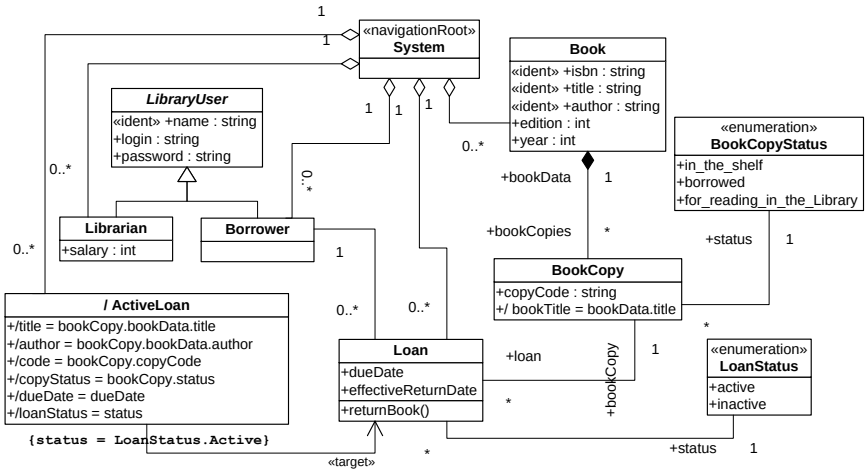

Fig. 5. Domain model (DM) for an example Library System.

4 Metamodel for Use Case Models

The metamodel for use case models, shown in Figures 6 and 7, specializes and extends the UML language unit for use cases [7].

In the approach proposed, the UCM is defined in close connection with the DM, to specify and organize the CRUD, user-defined or navigational operations over Base or Derived Entities that are available for each actor. The definition of a UCM also enables the use of several features, such as task-model-like relations, that permit a fine tuning of the interaction within a use case [8, 9], if the modeler wants to go deep in detailing a use case: enable, deactivate and choice. The data manipulated in each use case is typically determined by the domain entity (base or derived) and/or operation associated with it. Several constraints limit the types of use cases and UC relationships that can be defined [4].

The UCM metamodel extends UML and modifies standard elements UseCase, Extend and Operation. The UML UseCase has been added attributes that enable a smooth integration between a UCM and the respective DM.

A UseCase may identify an entity class (BaseEntity or DerivedEntity) from the DM. If a BaseEntity is identified, then it is possible to restrict the CRUD operations available, by associating only the allowed CRUD operations to the UseCase.

It has been defined a merge increment to the UML Extend class, for being possible to associate to it a link name and an aggregation operation. The association of an aggregation operation to an Extend only makes sense when the Extend is about an entity collection use case, in which case the associated aggregation operation must belong to the operations associated to the extended use case. Operations may be user-defined in an Action Semantics-based action language, or may be the basic CRUD instance operations that are defined by default in every BaseEntity.

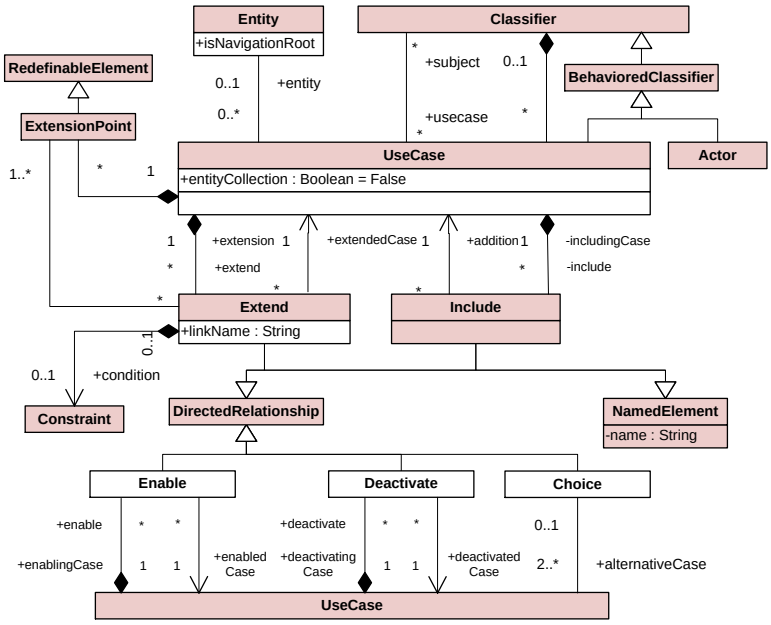

Fig. 6. Metamodel for Use Case Models.

One can distinguish two categories of use cases, which have been addressed in [2]:

- **Independent 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**, 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 optional or mandatory nature, respectively.

As can be seen in Fig. 7, a use case may be associated to an entity class in the DM.

If the modeler wants to reach a higher degree of detail, a use case may also be associated to an operation, entity attribute or a set of use case variables. This kind of use cases must be included in another use case, through an «include» relation, that "aggregates" them, sets the entity context and is able to bind variables to operations' parameters, or a specialization relation between use cases, that has the same effect as include, but lets the specialized UC inherit all inclusions, extensions and meta-attributes of the parent UC. Without losing the tight relation between the UCM and the DM, this enables the highest degree of flexibility in the UCM definition in order to better define what one wants to see generated in the UI model.

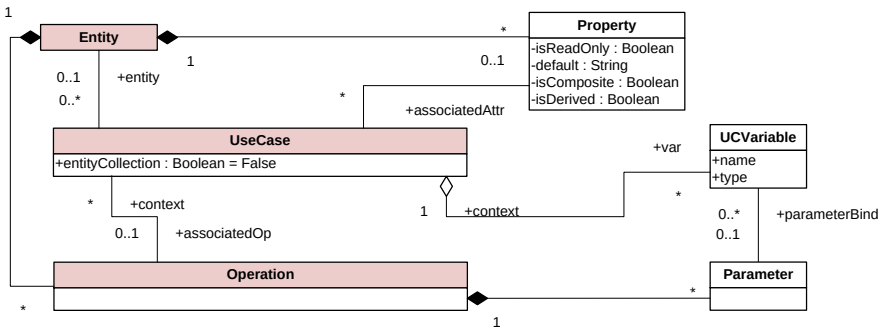

Fig. 7. Possible Use Case relations to Domain Model elements.

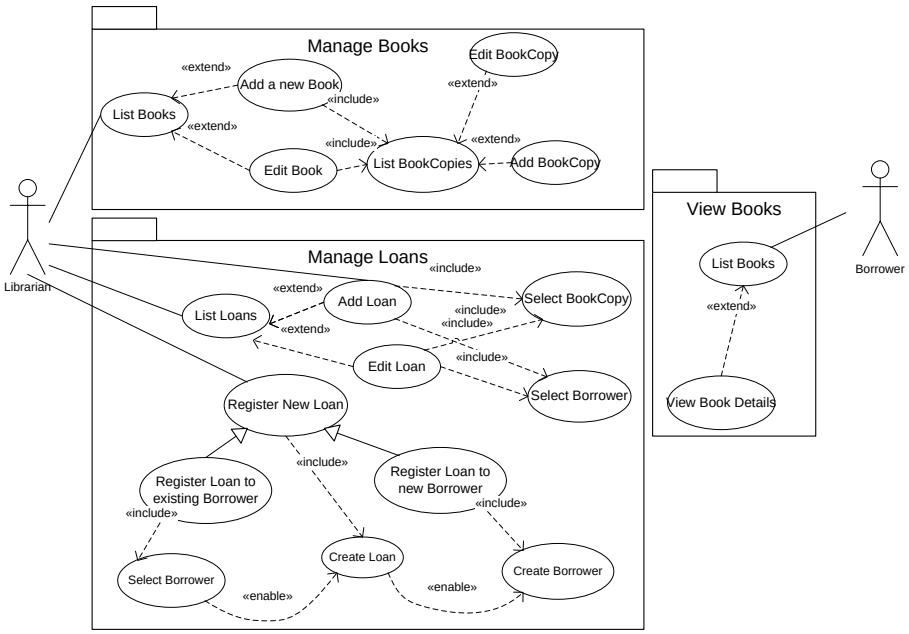

Fig. 8. Partial Use Case Model (UCM) for the LibrarySystem example.

Fig. 8 shows the UCM for the Library System example, which is fully integrated with the DM. Table 1 partially shows the entity types and operations associated (via tagged values) with some of the use cases. Fig. 8 includes UC “Register New Loan”, which is detailed by using specialized use cases and including, and enabling relations.

Table 1. Entities/operations associated (via tagged-values) with use cases in Fig. 8.

Use case	entity	entity Collection	associatedOp
List Books	Book	True	
Add a new Book	Book	False	Create
Edit Book	Book	False	Update, Delete
List BookCopies	BookCopy	True	
Add BookCopy	BookCopy	False	Create
Edit BookCopy	BookCopy	False	Update, Delete
View Book Details	Book	False	Retrieve
Select Borrower	Borrower	False	Update (link)
Create Borrower	Borrower	False	Create
Create Loan	Loan	False	Create

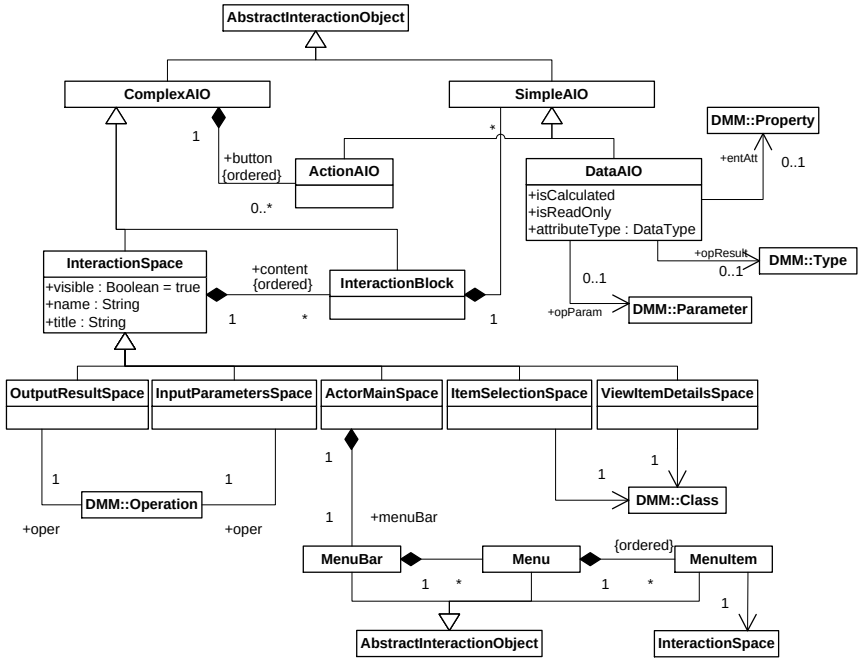

Fig. 9. Metamodel for User Interface Model.

5 Metamodel for User Interface Models

The proposed metamodel for User Interface models (UIM) is depicted in Figs. 9 and 10. The proposal focus on forms-based data-intensive applications, so the developed UIM metamodel contains elements for modeling forms and lists, and navigating

through them. The UIMM imports elements defined in the DMM, in order to guarantee the whole system model consistency, and together have all the information needed to generate an executable prototype.

The top level element from which every element in the UIMM inherits from is AbstractInteractionObject, or AIO. There are two types of AIO: ComplexAIO and SimpleAIO. The first models elements that contain other elements, and the latter models simple elementary objects used within complexAIO elements.

An InteractionSpace (IS) represents an abstract object that, at PIM level, is a UI container where interaction occurs. An IS is composed of InteractionBlocks, which, in turn, are made of SimpleAIOs, like DataAIOs, that are typically associated to entity properties in the DM, or ActionAIOs. These latter may be DomainOperations, like the call of a CRUD or user defined operation, operations for Navigation between InteractionSpaces, or UIOperations, that allow actions over UI elements.

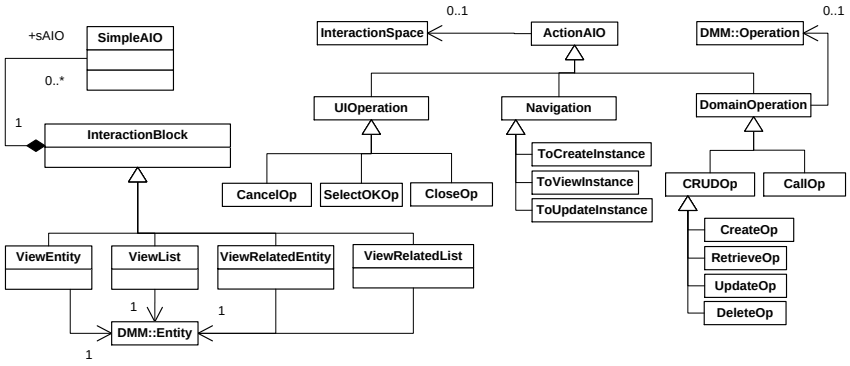

Fig. 10. UIM Metamodel parts for the InteractionBlock subtree, in the left, and the ActionAIO subtree, at the right.

6 Model Transformation Rules

According to the presented development process (see section 2) an automatic model transformation process is able to generate a UIM from the DM and UCM. The structural information of the UIM is derived from the DM, although its presentation may be restricted for some users according to what may be defined in the UCM. Although implemented imperatively in the proof-of-concept tool, the transformation rules were defined declaratively through LHS-RHS relations, in which the RHS defines the elements being generated or modified in the target model when a pattern matching is verified in the LHS. Due to space limitations, the mapping rules that drive the transformation process from the DM and UCM to the UIM cannot be fully presented in this paper. A complete presentation can be found in [4].

In what respects to structural UI information, a base domain entity is by default mapped to an InteractionSpace with a ViewEntity block, with a DataAIO for each attribute and ActionAIOs for the CRUD operations, depending on the context

(creating a new instance or editing an existing instance). Inheritance is treated by creating a DataAIO for each inherited attribute in the ViewEntity generated for the specialized entity.

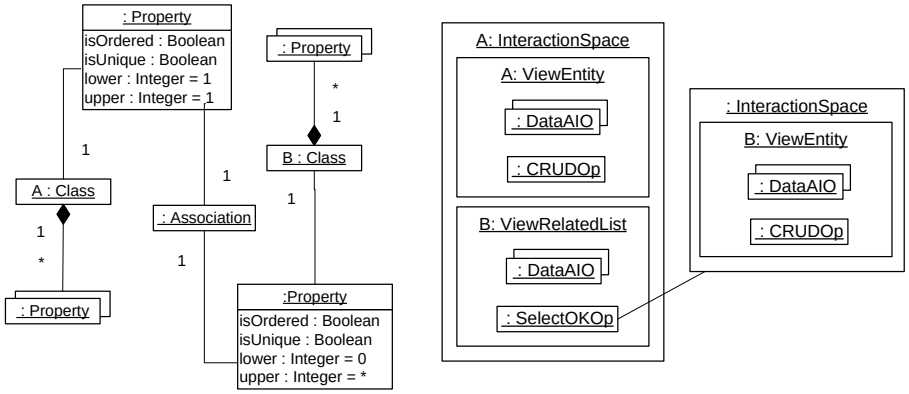

Fig. 11. A sample DM metamodel instance (abstract syntax), in the left, and a partial correspondence to a UIM metamodel instance.

To-many relations (associations, aggregations or compositions) are mapped to a ViewRelatedList block in the InteractionSpace generated for the source entity, with a list of DataAIOs, one for each identifying attribute of the related instances of the target class, and ActionAIOs for adding new instances and for editing or removing the currently selected instance.

To-one relations are mapped to a ViewRelatedEntity block in the interaction space generated for the source class, with a DataAIO for each identifying attribute of the related instance of the target class. If the UIM is being generated solely from the DM, and if the related instance is not fixed by the navigation path followed so forth, then a Navigation AIO is also generated for selecting the related instance.

Within ViewList and ViewRelatedList blocks, DataAIO objects will correspond to table columns in concrete representations, while in other kinds of interaction blocks they will correspond, for instance, to labels and text fields.

Derived attributes map to output-only DataAIOs, and default values are mapped to initial values in DataAIOs. Each derived entity (view) is mapped to an interaction space with a ViewEntity having an input/output DataAIO for each attribute of the target class, an output-only DataAIO for each derived attribute, and ActionAIOs for the CRUD operations over the target class.

In what respects to UI-functionality related information, from the UCM, an actor is mapped into an ActorMainSpace, where the actor starts its application usage experience. A Use Case Package is mapped to a menu in the actor's main space, with a menu item for each UC within the package that is directly linked to the actor.

A use case (UC) with attribute entityCollection set to true (UC of type *List Entity* or *List Related Entity*) is mapped to an interaction space with a ViewList or a ViewRelatedList, depending if it is an independent or a dependent UC, displaying the full list of instances of the associated entity, with ActionAIOs for the allowed operations (according to the dependent use cases).

An extension UC associated to an entity that is the "many" side of a relation with the entity associated to the extended UC (UC of type *Select Related Entity*) is mapped to an interaction space with a ViewList displaying the list of candidate instances, with ActionAIOs for selecting one instance.

A UC with attribute entityCollection set to false (UC of type *CRUD Entity* or *CRUD Related Entity*) is mapped to an interaction space with a ViewEntity or a ViewRelatedEntity block displaying the object attribute values, through DataAIOs, with ActionAIOs corresponding to the CRUD operations allowed. In the case of a related instance, the showed «ident» attributes of the source object cannot be edited.

A UC associated to a user-defined operation (UC of type *Call User-Defined Operation*) generates a CallOp within the ViewEntity corresponding to the entity where the operation is defined. Blocks for entering the input parameters and displaying the result, in case they exist, are also generated.

A Navigation ActionAIO is typically generated from an Extend or an Include relationship.

Fig. 12. Excerpt of the prototype generated from the LibrarySystem.

Fig. 12 shows part of the prototype generated for the LibrarySystem, showing the UI flow of a Librarian actor executing use cases List Books → Edit Book.

7 Related Work

Few approaches in the literature allow a model-to-model generation of a UIM and prototype, within a MDD setting. The XIS method [10, 11], like the OO-Method [12] and the ZOOM approach [13] are able to produce a fully functional (executable) application, but the demanded input models are time consuming and arduous to build.

XIS allows two approaches to interactive systems generation: a dummy approach that demands the full specification of a DM, an actors' model, and a UIM; and a smart approach, which enables the derivation of the UIM 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 [10, 11]. Like in the XIS smart approach, in our approach the modeler must attach to each use case an Entity (base or derived) from the DM. The difference is that, in our approach, relations between entities are inferred from the DM, 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 is done, within our approach, in the UCM by modeling or not related use cases associated to related entities for navigating through the relations.

Similarly to XIS and the OO-Method, in our approach CRUD operations are predefined, but their default behavior may be modified through domain triggers. User defined operations are not possible to specify in the XIS approach. In the OO-Method, user defined operations (services and transactions) can be specified by using formal language OASIS or, in a limited form, by specifying the way each service changes the object state, categorizing each attribute [12]. 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. In our approach user defined operations may be specified using an UML Action Semantics-based language.

The ZOOM approach models a system by building a graphical model, which is then translated to the ZOOM language [13]. The models that are demanded by the approach, in order to automatically generate an executable application, are: a structural model, which must contain all the classes of the application (including design classes); 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. [14] and Martinez et al. [15] approaches generate 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 or 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.

8 Conclusions and Future Work

This paper addresses an approach to MDD comprising a development process and a UML-aligned model architecture that enable a gradual approximation to the final application by deriving a default UIM from early DM and UCM and an application prototype from these three system model views. On each iteration the system models are further refined in order to accommodate all the requirements and to generate a suitable UIM. UI look&feel is achieved by modifying the UIM, without detaching it from the related DM, and by applying stylesheets to the generated UI code.

The approach proposed combines advantages of the state-of-art approaches and adds a few own contributions. The main distinguishing point is that it doesn't demand

a UIM for UI generation; instead, it is able to generate a UIM and executable prototype from the DM alone or the UCM also. Other points are that it takes advantage of OCL invariants and preconditions to generate validation routines in the executable prototype; adds use case relations based on constructs typically found in task-models; allows defining triggers activated by CRUD-operations' invocation or by a state condition holding; makes use of an actions language to specify triggers and class operations [4].

The presented metamodels enable a complete and rigorous modeling of a data-intensive form-based system. Being this the case of most of the business applications, this approach enables full final-code generation, provided that target platforms and architecture may influence the selection of a model-to-code generator.

Future evolutions may include: the development of a complete support tool that may integrate with existing MDA or UML diagramming tools; or, the opportunity to specify the target architectures and feed them to a generic model-to-code generator.

References

1. Frankel, D.S.: Model Driven Architecture - Applying MDA to Enterprise Computing. Wiley Publishing, Inc., Indianapolis, Indiana, 2003.
2. Cruz, A.M.R., Faria, J.P.. Automatic generation of user interface models and prototypes from domain and use case models. In *Proceedings of the 4th ICSOFT* (ICSoft 2009) , vol. 1, pp 169-176, Sofia, Bulgaria, INSTICC Press, July 2009.
3. Cruz, A.M.R., Faria, J.P.. Automatic Generation of Interactive Prototypes for Domain Model Validation. In *Proceedings of the 3rd ICSOFT* (ICSoft 2008) , vol. SE/GSDCA/MUSE, pp 206-213, Porto, Portugal, INSTICC Press, July 2008.
4. Cruz, A.M.R. Automatic generation of user interfaces from rigorous domain and use case models. PhD thesis (to be published). F.E.U.P., University of Porto, Portugal, 2010.
5. Jacobson, I., Booch, G., Rumbaugh, J.. The Unified Software Development Process, Addison Wesley, 1998.
6. OMG: Unified Modeling Language (OMG UML) Infrastructure. February 2009.
7. OMG: Unified Modeling Language (OMG UML) Superstructure. February 2009.
8. Paternó, F., Mancini, C., Meniconi, S.. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In *Proceedings of the IFIP TC13 Int'l Conf. on HCI*, INTERACT '97, Chapman & Hall, Ltd., 362-369, 1997.
9. Paternó, F.. Task Models in Interactive Software Systems. In *Handbook of Software Engineering and Knowledge Engineering, vol I.* World Scientific Publ., pp. 817–835, 2001.
10. Silva, A.: The XIS approach and principles. In Society, I.C., ed.: Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture", 2003.
11. Silva, A. R., Saraiva, J., Silva, R., Martins, C., XIS - UML Profile for eXtreme Modeling Interactive Systems. *4th International Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES 2007), IEEE Computer Society*, 2007.
12. Pastor, O., Molina, J.C.: Model-Driven Architecture in Practice. *Springer-Verlag*, 2007.
13. Jia, X., Steele, A., Qin, L., Liu, H., Jones, C.: Executable visual software modeling the ZOOM approach. Software Quality Control 15(1), pp 27-51, 2007.
14. Elkoutbi, M., Khriss, I., Keller, R.: Automated prototyping of user interfaces based on UML scenarios. Journal of Automated Software Engineering 13(1), pp. 5-40, January 2006.
15. Martinez, A., Estrada, H., Sanchez, J., Pastor, O.: From early requirements to user interface prototyping: A methodological approach. In: Proceedings of ASE 2002, pp. 257-260, 2002.