Using UML Profiles to Interchange DSML and UML Models

A Proposal for MDD Approaches

Giovanni Giachetti, Beatriz Marín, Oscar Pastor
Centro de Investigación en Métodos de Producción de Software
Universidad Politécnica de Valencia
Camino de Vera s/n
46022 Valencia, Spain
{ggiachetti, bmarin, opastor}@pros.upv.es

***Abstract*—A key requirement for MDD solutions is to have a modeling language that allows the correct representation of conceptual models. Nowadays, there are two options that are the most widely used for the definition of these modeling languages: 1) the specification of a domain-specific modeling language (DSML) or 2) the customization of UML. In practice, these two modeling alternatives are viewed as opposite solutions. However, since both alternatives provide benefits for the application of MDD solutions, in this paper, we present a proposal that uses UML profile extension mechanisms to interchange modeling information between DSML-based models and UML models. This proposal shows how these two modeling alternatives can be integrated in a unique MDD solution.**

UML profile; DSML; UML; MDD

I. INTRODUCTION

One of the most important concerns when elaborating an MDD solution [26] is the specification of a modeling language that allows the required software products to be represented at the conceptual level without ambiguity. Among the different choices that exist for the definition of an adequate modeling language, there are two alternatives that appear to be the most suitable. The first of these is the creation of a specific language that is tailor-made for the MDD approach. This language is called the *Domain-Specific Modeling Language* (DSML) [25]. The second alternative is the customization of UML with the required semantics [27] by means of one of the existent UML extension mechanisms [2].

An analysis of the first modeling alternative indicates that a DSML provides a precise characterization of the conceptual constructs required by MDD approaches. Furthermore, the number of conceptual constructs required by a MDD proposal is generally smaller than the number of conceptual constructs that are defined in the UML Specification [22]. For this reason, the implementation of specific MDD tools (such as model compilers or documentation tools) is easier to perform for a DSML than for UML. At the same time, the smaller size of a DSML in relation to UML facilitates the implementation of the specific model editors that are required for the definition of DSML-based models. These model editors can provide improved modeling features that are in accordance with the development domain. However, it is important to note that the implementation and maintenance of these specific editors involve an extra effort when putting an MDD approach in practice [27].

The second alternative for the specification of an adequate modeling language is the use of UML, which is a widely known modeling language that has a lot of support tools. However, it is also true that the semantics of certain UML conceptual constructs does not provide enough precision for its application in effective MDD processes. This lack of precision can be easily perceived in the *Semantic Extension Points* defined in the UML specification [22], where different semantic representations for one conceptual construct or the lack of definition of the appropriate semantics can be found. An example of this is the semantic extension point related to the UML association, which state that "The order and way in which part instances in a composite are created is not defined" [8].

In order to introduce the required semantic precision into UML, there are different extension mechanisms that can be used [2]. One of them, the UML profile extension mechanism, is the most suitable extension alternative because it is part of the UML standard and, hence, the extensions defined with a UML profile can be supported by UML-based tools. Therefore, existent MDD technologies based on UML (such as requirement traceability tools or cost estimation tools) can be reused by other MDD solutions that also use UML as the base modeling language. In addition the UML profile extension mechanism allows the existent UML editors to be used thereby reducing the costs of implementing specific model editors.

However, since UML is a general purpose modeling language, the modeling facilities that the existent UML editors provide may not be the most appropriate to perform specific modeling tasks related to MDD approaches. Furthermore, the extension capabilities of the UML profile present limitations that in some cases might prevent a correct representation of all the modeling needs that are required by MDD proposals.

9781-4244-2865-6/09/$25.00 ©2009 IEEE

After analyzing these two modeling alternatives (UML and DSMLs), an interesting modeling approach would be to provide a hybrid modeling schema that integrates both alternatives. This integration can be obtained by means of the transparent interchange of UML models and DSML models. Thus, it would be possible to take advantage of the existent UML tools for those models that can be represented by means of UML and only to implement specific tools for those models that require more complex modeling capabilities. It would also be possible to implement specific DSML-based tools for those features that are outside of the scope of existent UML tools.

This paper presents a proposal to obtain this hybrid modeling schema. This proposal is based on the automatic generation of a UML profile from a DSML metamodel. All the information required to interchange DSML models and UML models is obtained from this UML profile generation. Finally the application of this interchange proposal into a specific MDD approach is presented by means of a brief example.

The rest of the paper is organized as follows: Section 2 presents the background related to UML profiles and DSMLs. Section 3 introduces our proposal for the UML profile generation process. Section 4 details how the results obtained in the UML profile generation are used for the model interchange. Section 5 presents the application example of the interchange proposal. Section 6 presents a discussion about the related works, and finally, Section 7 presents our conclusions and further work.

II. BACKGROUND

This section briefly introduces the aspects that are relevant to the definition of DSMLs and UML profiles. This background is specially focused on those aspects that can be used to perform an interchange between DSML-based models and UML models that are extended with UML profiles.

A. Domain-Specific Modeling Languages

A *Domain-Specific Modeling Language* (DSML) represents the semantics of the constructs required for the definition of the conceptual models involved in a MDD solution. For the construction of a DSML, a common strategy is to define a metamodel to represent the abstract syntax of the required conceptual constructs [13]. For the elaboration of this *DSML Metamodel*, one of the most interesting alternatives is the use of the EMOF standard defined by OMG [20]. EMOF is the essential set of metamodeling constructs, which is defined within the MOF specification [19]. The benefit of using this standard for the definition of metamodels is that it facilitates the interchange and validation of the metamodels and support from existent metamodeling tools, such as EMF project [5], can be obtained. EMF is an Eclipse project that provides an open-source implementation of the EMOF standard, which is called *Ecore*. In addition, by means of tools like Eclipse GMF [6] and Eclipse ATL [4], specific model editors and model transformations can be defined over Ecore metamodels.

B. UML Profiles

The UML profile extension mechanism is defined inside of the UML Infrastructure [21]. It defines the mechanisms used to adapt existing metamodels to specific platforms, domains, business objects, or software process modeling. Since this extension mechanism is a part of the UML standard, it can be supported by UML tools. This feature is one of the main advantages of the UML profile over other UML customization mechanisms [2], which are not part of the UML standard and, hence, they are not supported by UML tools.

A UML profile is represented as a UML package that is stereotyped with the tag *<<profile>>*. It has three main constructs for the definition of the required extensions: stereotypes, tagged values and OCL rules.

Figure 2 shows a brief example of a UML profile. This UML profile has the stereotype *IdClass* that extends the metaclass *Class* with an identifier, which is specified by means of the tagged value *identifier*. The stereotype *IdClass* also has an OCL rule that indicates that the assignment of a value for the identifier is mandatory.

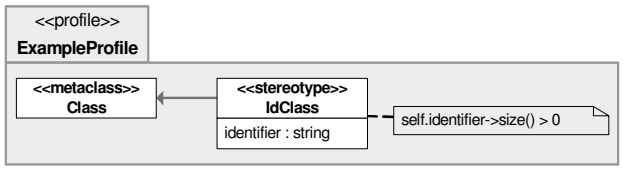


Figure 1. Example of a UML profile

Generally speaking, UML profiles are manually elaborated without a well-defined process. This situation is motivated by the lack of a standard that specifies how the UML extensions must be defined [8]. For this reason, many of the existent UML profiles are invalid or of poor quality [27]. To avoid this situation, some works propose a more methodical solution that consists in the definition of a UML profile from the metamodel that describes the conceptual constructs required by MDD approaches. In other words, the UML profile is generated from the DSML metamodel [27]. This UML profile generation schema is based on the identification of equivalences (mapping) between the DSML metamodel and the UML metamodel. Later, the identified equivalences are used to guide the correct definition of the required extension over UML.

Certain proposals state that these equivalences can be used to partially automate the UML profile generation [15][28]. However, these proposals cannot provide a totally automated solution for the generation of a complete UML profile. This occurs because, in real MDD approaches, certain structural differences between the DSML metamodel and the UML metamodel may appear. This prevents the automated identification of all the extensions that must be performed in UML.

The proposal presented in [12] defines a solution to solve these structural differences in order to obtain an adequate input for an automated UML profile generation. Therefore, considering that the UML profile is generated from the DSML metamodel, the information of the equivalences between the generated UML profile and the DSML can also be obtained from this generation. With this information, UML models that are extended with the generated UML profile can be

automatically transformed into the equivalent DSML metamodels and vice versa. The interchange proposal presented in this paper is based on this idea. The UML profile generation process used for the application of our interchange proposal is presented in the following section.

III. A UML PROFILE GENERATION PROCESS

Our interchange proposal is based on a process that allows the automatic generation of a UML profile (see Figure 2). This process uses the DSML metamodel related to a MDD approach as input. In the definition of this process, different works have been considered. Some of these works are: definition of profiles using DSML metamodels [9][15][27][28], correct use of metamodels in software engineering [14], UML profile implementations [18], interchange between UML and DSMLs [1][17], and new UML profile features introduced in UML [21].

The proposed UML profile generation process can be used in those MDD approaches where the required conceptual constructs can be represented starting from existent UML constructs. This constraint comes from the limitation of the extensions that can be performed with a UML profile, which cannot change the reference metamodel. The conceptual constructs that conform the UML standard are presented in the UML Superstructure [22].

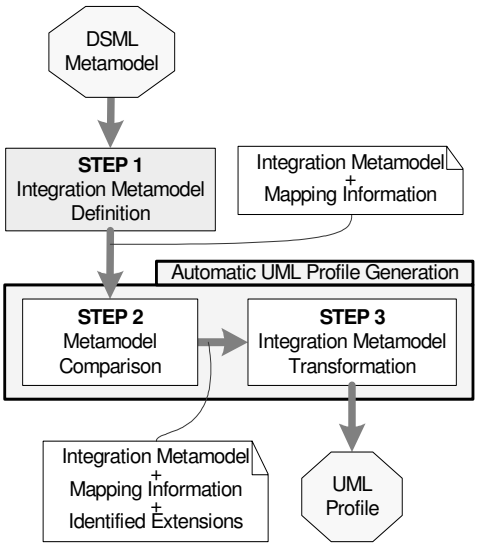


Figure 2. Schema of the UML profile generation process

The process presented in Figure 2 is composed of three steps:

- *Step 1*: The first step corresponds to the generation of a special metamodel from the DSML metamodel. This special metamodel, which is called the *Integration Metamodel*, provides an adequate input for the automatic UML profile generation.
- *Step 2*: In the second step, a comparison between the Integration Metamodel and the UML metamodel is performed. This comparison identifies the required UML extensions.
- *Step 3*: In the third and last step of the process, the Integration Metamodel generated in *Step 1* is transformed into the final UML profile.

In order to understand how the interchange between UML and DSML models can be performed through this UML profile generation process, the three steps of this process are detailed below.

A. Step 1: Integration Metamodel Definition

The Integration Metamodel is the proposal presented in [12] to improve the automatic generation of a UML profile. This metamodel allows the abstract syntax represented in a DSML metamodel to be automatically integrated into UML. This metamodel is defined from the DSML metamodel, and it represents the same abstract syntax of the original metamodel. The main difference between the Integration Metamodel and the DSML metamodel is its structure since it is defined to obtain a mapping with the UML metamodel, which allows the automatic identification of the required UML extensions.

The Integration Metamodel has the following features:

- It is defined according to the EMOF modeling capabilities, which are defined in the MOF (Meta Object Facility) specification [19].
- It is mapped to the UML metamodel taking into account: Classes, Attributes, Associations, Enumerations, Enumeration Literals, and Data Types.
- All the classes from the Integration Metamodel are mapped to classes of the UML metamodel. This assures that the conceptual constructs of the DSML can be represented from the conceptual constructs of UML.
- The mapping information is included inside the Integration Metamodel definition by using a special stereotype. Thus, all the information needed to generate a UML Profile is integrated in a unique XMI file that is defined according to the OMG Standards.

During the Integration Metamodel definition, the original structure of the DSML metamodel may be redefined. This redefinition is performed without altering the abstract syntax represented in the DSML metamodel. This is illustrated by the DSML metamodel presented in Figure 3.

Figure 3 shows a DSML metamodel that represents a binary association between classes. In this metamodel, the class (metaclass) *DMClass* represents classes of a model, and the class *DMAssociation* represents binary associations. The class *DMAssociation* has two associations that represent the participant classes, and four attributes to specify the lower and upper bound of each association end. In these classes, two attributes for the generic representation of properties are defined. These attributes are *newAttr1* for the class *DMClass* and *newAttr2* for the class *DMAssociation*.

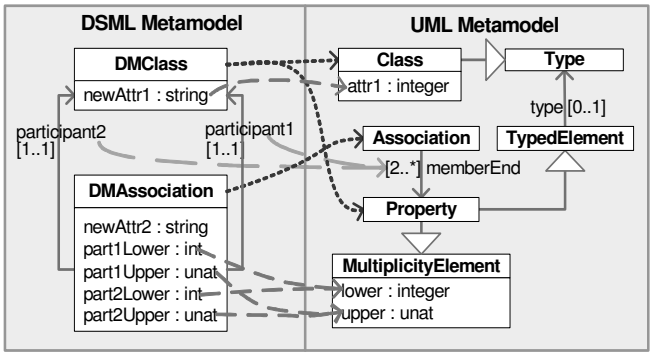


Figure 3. DSML metamodel example

As Figure 3 shows, the DSML metamodel must be mapped to the corresponding classes of the UML metamodel. This mapping defines the equivalences that exist between the two metamodels. For instance, the mapping of the example (see Figure 3) indicates that the *DMClass* is mapped to the classes *Class* and *Property*. The mapping between *DMClass* and *Class* is clear because both metaclasses represent classes of a model. The mapping between *DMClass* and *Property* is defined because, in UML, the participant classes of an association are represented by object-valued properties, where the types of these properties correspond to the related classes.

However, the DSML metamodel and the defined mapping information are not enough to automatically identify the extensions that are required to generate the corresponding UML profile. An example of this can be observed in the mapping that is defined for the class *DMClass*. With this mapping, it is impossible to know if the attribute *newAttr1* must be considered to extend the class *Property* by means of a tagged valued defined in the generated UML profile. Or by contrast, it is only an attribute related to the syntax of a class and must not be considered to extend the class *Property* for the correct representation of an association end. In order to obtain an adequate mapping, an Integration Metamodel is defined from the DSML metamodel according to the systematic approach presented in [12]. Figure 4 shows the Integration Metamodel obtained.

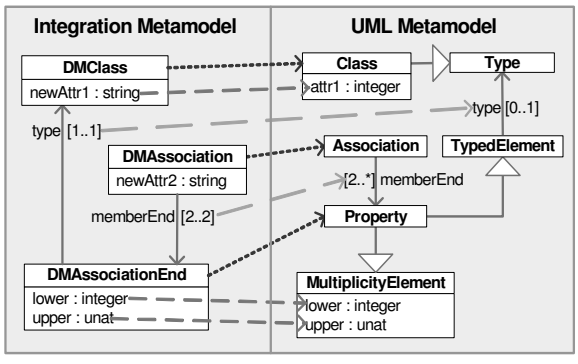


Figure 4. Integration Metamodel for the example

In the Integration Metamodel presented in Figure 4, it can be observed that the conflicts related to the mapping of the class *DMClass* are solved by the definition of a new class called *DMAssociationEnd*, which is mapped to the UML class *Property*. Thus, with the mapping obtained for the Integration Metamodel, it is clear that the attribute *newAttr1* is only involved in the representation of classes and it must not be considered to extend the class *Property*.

The new class *DMAssociationEnd* represents the participant classes of the association identified by means of the association *type*. The association *memberEnd* with cardinality [2..2] assures that an association only has two participants. Thus, the resultant Integration Metamodel represents the same abstract syntax as the DSML metamodel.

In addition, the mapping information between the Integration Metamodel and the original DSML metamodel is also obtained in the definition of the Integration Metamodel. This information is used in the interchange of UML and DSML models and to assure that the resultant Integration Metamodel represents the same syntax as the original DSML metamodel.

The mapping information obtained for the example is presented in Table I. This Table shows that the first and second instances of the association *memberEnd* represent the first and second participant classes of an association.

TABLE I. MAPPING INFORMATION BETWEEN THE DSML METAMODEL AND THE INTEGRATION METAMODEL.

DSML Metamodel	Integration Metamodel
DMClass	DMClass
.newAttr1	.newAttr1
DMAssociation	DMAssociation
.newAttr2	.newAttr2
.part1Lower	.memberEnd(1).lower
.part1Upper	.memberEnd(1).upper
.part2Lower	.memberEnd(2).lower
.part2.Upper	.memberEnd(2).upper

The obtained Integration metamodel is used in the automatic UML profile generation, which corresponds to the second and third steps of the UML profile generation process.

B. Step 2: Metamodel Comparison

The second step allows the automatic identification of the of required UML extensions through a comparison between the Integration Metamodel and the UML metamodel. To perform this comparison, the mapping information defined in the Integration Metamodel is used.

The automatic comparison between the Integration Metamodel and the UML Superstructure considers:

- The identification of *new elements*, which are the elements from the Integration Metamodel that are not equivalent (mapped) to UML elements. These elements can be attributes, associations, enumerations, literal values, and data types.
- The identification of *differences in type or cardinality* of *equivalent properties* (attributes and associations).

The result of the comparison obtained for the example is presented in Table II. In this table, the *Difference* column

shows what the differences are by indicating (when necessary) the values for the Integration Metamodel element (*I.M.*) and the UML element (*UML*).

TABLE II. METAMODEL COMPARISON RESULTS

Integration Metamodel	Difference
DMClass.newAttr1	Different type: I.M. = string; UML = integer
DMAssociation.memberEnd	Different upper bound: I.M. = 2; UML = *
DMAssociation.newAttr2	New attribute
DMAssociationEnd.type	Different lower bound: I.M. = 1; UML = 0 Different type: I.M. = DMClass; UML = Type

C. Step 3: Integration Metamodel Transformation

The transformation of the Integration Metamodel into the corresponding UML profile is driven by a set of transformation rules [11]. The features of these transformation rules that are relevant for the interchange proposal are presented below:

- Each class of the Integration Metamodel is represented by a stereotype of the generated UML profile.
- The equivalent elements are represented with the corresponding UML elements, according to the mapping defined in the Integration Metamodel. If there are differences in equivalent elements according to the results obtained in the metamodel comparison (Step 2), then these differences are managed with OCL constraints. However, there are equivalent elements whose differences cannot be managed with OCL constraints. In these cases, an element defined in the UML profile replaces the original UML element.
- The new elements identified in the metamodel comparison are directly represented in the UML profile.

Figure 5 shows the UML profile obtained from the Integration Metamodel of the example. This figure presents a simplified version of the UML profile, which is really generated by the transformations rules. This simplification eliminates a set of validation constraints that are not relevant for understanding the interchange proposal.

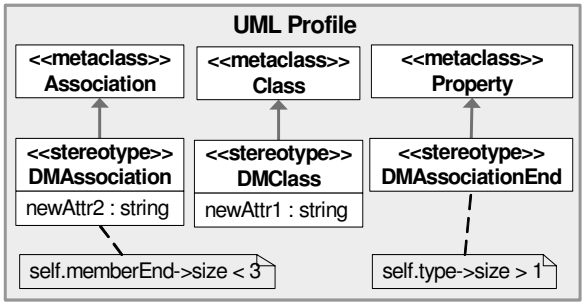


Figure 5. Generated UML profile

Figure 5 also shows that the classes *DMAssociation*, *DMClass*, and *DMAssociationEnd* are represented by three stereotypes with the same names. The equivalent association *DMAssociation.memberEnd* has a direct representation in the UML association (*Association.memberEnd*) according to the mapping defined in the Integration Metamodel. In this case, an OCL constraint is defined to manage the upper bound difference. A similar situation can be observed in *DMAssociationEnd.type*. However, for the attribute *DMClass.newAttr1*, the type difference cannot be managed with an OCL constraint. To solve this, a tagged value (with the same name) is defined in the stereotype *DMClass*. This tagged value replaces the original UML attribute. Finally, the new attribute *DMAssociation.newAttr2* is represented by means of a tagged value that is defined in the stereotype *DMAssociation*.

An interesting characteristic that can be observed in this transformation example is that each element of the Integration Metamodel can be represented by an element of the UML metamodel or by an element of the UML profile. Therefore, it is possible to transform a UML model that is extended with the generated UML profile into an equivalent model defined according to the Integration Metamodel constructs. However, to achieve this transformation, the equivalences (mapping) between the elements of these two metamodels must be known.

Therefore, the transformation rules not only generate the UML profile, they also generate the mapping between the Integration Metamodel and the UML metamodel that is extended with the obtained UML profile.

Figure 6 shows graphically the mapping obtained between the Integration Metamodel and the UML profile generated in the example.

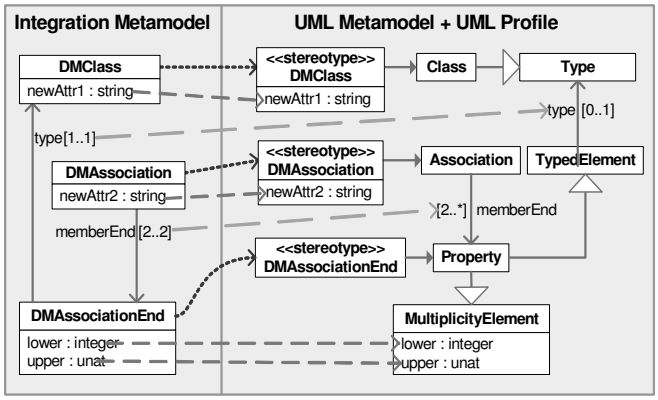


Figure 6. Mapping for the generated UML profile

The mapping between the Integration Metamodel and the generated UML profile is generated according to the following guidelines:

- Each class from the Integration Metamodel is mapped to the stereotype generated from this class.
- Each equivalent element of the Integration Metamodel is mapped to the corresponding UML elements, except for those elements with differences that cannot be

represented by means of OCL rules. In that case, the equivalent element is mapped to the element of the UML profile that is defined to replace the original UML element.

- Each new element of the Integration Metamodel is mapped to the corresponding element of the generated UML profile.

The next section shows how the mapping information obtained during the UML profile generation process can be used to interchange UML and DSML models.

IV. THE DSML AND UML INTERCHANGE PROPOSAL

The interchange proposal requires two model transformations (see Figure 7) that are performed using the mapping information obtained in the UML profile generation process.

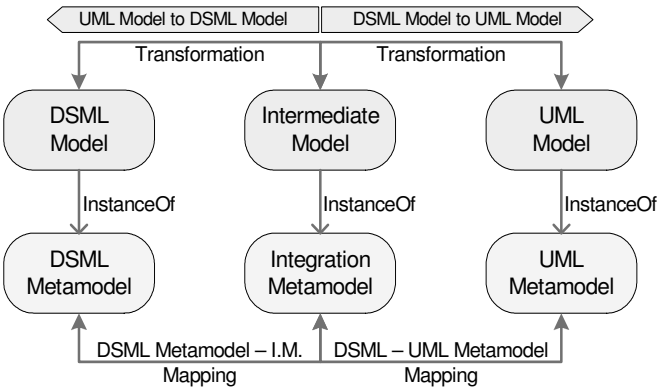


Figure 7. Interchange Between DSML and UML models

With this proposal, it is possible to generate DSML models from UML models that are extended with the generated UML profile, and vice versa. For instance, to obtain a DSML model from a UML model, the first transformation generates an *intermediate model* from the UML model. This intermediate model is an instance of the Integration Metamodel. This first transformation requires the mapping between the Integration Metamodel and the generated UML profile. The second transformation generates the final DSML model from the intermediate model obtained in the first transformation. This second transformation requires the mapping between the Integration Metamodel and the DSML metamodel.

Figure 8 shows an interchange example that is based on the metamodels and mappings obtained for the example presented in the UML profile generation process (see section III). This example represents an association *many-to-many* between the classes *Passenger* and *Flight*. The corresponding model for this association is represented for the UML model that is extended with the generated UML profile, the Intermediate model, and the DSML model. Figure 8 also shows how can be performed the transformation from one model to other by means of the correspondences between the elements of each model, which are defined according to the mapping obtained in the UML profile generation. In this figure, the name of the association ends in the UML model has been omitted because it is not relevant for the model interchange.

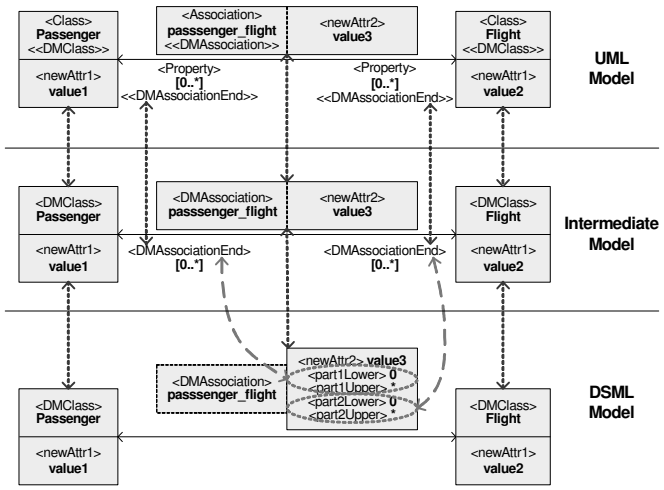


Figure 8. Example models obtained in the interchange proposal

A recommended strategy for the implementation of the required model transformations is the use of model-to-model transformation technologies such as QVT or ATL. Thus, an interesting open-source implementation alternative is the Eclipse ATL project, which performs model transformations by means of a mapping that is defined between Ecore metamodels [5]. In addition, the Eclipse ATL project also provide support to perform transformations over UML models that are defined with the Eclipse UML2 project [7].

The proposal of Abouzahra et al. presented in [1] shows a practical approach based on ATL that can be useful to learn how transform a UML model extended with a UML profile into another target model. To perform this transformation, the manual definition of a mapping between the UML profile and the metamodel related to the target model is required. In our proposal, this mapping corresponds to the mapping between the UML profile and the Integration metamodel, which is automatically obtained. Thus, the Abouzahra et al. proposal could be an interesting open-source solution to implement the transformations between the UML model and the intermediate model that is presented in the interchange proposal. However, it is still necessary to implement the transformations in order to obtain the DSML model from the intermediate model, and vice versa.

Nevertheless, it is important to remark that for the implementation of the interchange proposal can be used other transformation alternatives, which are different than model-to-model transformations, for instance, by means of XSLT transformations.

The next section shows how has been applied the interchange proposal in an industrial MDD solution, and presents a brief application example of this implementation.

V. APPLIYING THE INTERCHANGE PROPOSAL

The proposed interchange proposal has been applied to an industrial MDD solution that is called *OlivaNova* (*the Programming Machine*) [3][24]. Olivanova corresponds to the industrial implementation of the OO-Method approach [23]. Olivanova has been selected because it already has a suite of MDD tools that are based on the OO-Method DSML. Thus, by applying our interchange proposal to Olivanova, UML-based tools can be integrated with the existent Olivanova technology.

In the suite of Olivanova tools, there are two tools that are oriented to interchange UML and OO-Method models. These tools are the OO-Method XMI importer, which transforms an UML model into an OO-Method model, and the OO-Method XMI exporter, which transforms an OO-Method model into a UML model [17]. The transformations performed by these tools are implemented through XSLT transformations.

However, the interchange provided by these interchange tools is limited by the UML modeling capabilities because the UML constructs do not provide all the precision required by the OO-Method conceptual model. In order to solve this problem, a UML profile that extends UML with the precision required by OO-Method has been defined using the generation process presented in Section 3. The mapping information obtained during the generation of the UML profile has been used to extend the OO-Method interchange tools in order to improve the interchange between UML and OO-Method models. In order to perform this improvement, the transformations required by the interchange proposal are implemented by means of XSLT transformations. This implementation decision has been taken because the OO-method interchange tools are originally based on XSLT.

The implementation schema that is used to apply the interchange proposal in the Olivanova Technology is presented in Figure 9. According to this schema, the OO-Method metamodel is used as input for the UML profile generation process.

Figure 9 shows that the UML profile generation process also generates the mappings required to perform the model transformation involved in the interchange of UML and OO-Method models. It is important to note that the mapping between the Integration Metamodel and the UML metamodel is automatically generated during the automatic UML profile generation.

In the importation process, the XMI importer tool uses as input the XMI definition of the UML model (extended with the generated UML profile) and generates as output an equivalent OO-Method model, which is represented in XML format according to a specific DTD. The exportation process performs the opposite transformation, it takes as input the XML representation of an OO-Method model and generates as output an equivalent UML model. Therefore, the XMI definition of the UML model can be used by UML-based tools, and the XML definition of the OO-Method object model can be used by the different Olivanova tools and by the Olivanova model compilers.

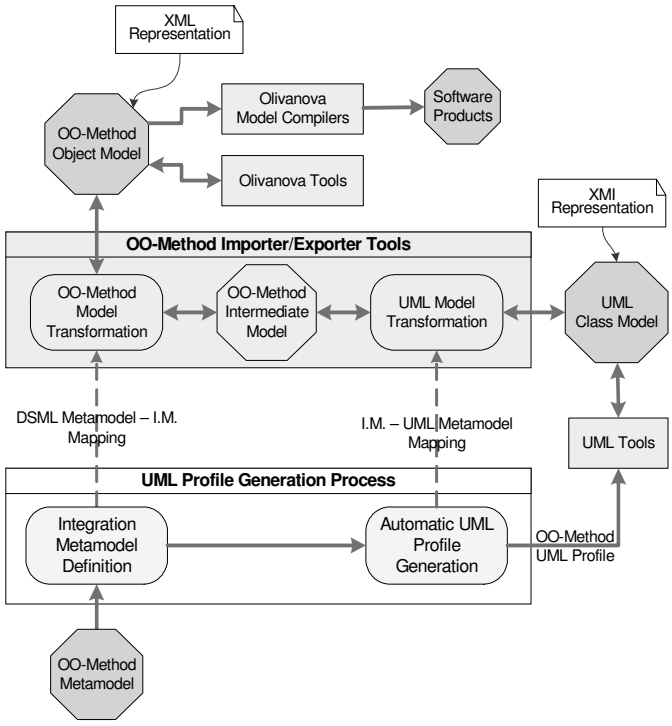


Figure 9. Schema for the implementation of the interchange proposal in OO-Method

The interchange between OO-Method and UML is centered on the OO-Method object model, which is similar to the UML class model. Therefore, the interchange tools are focused on transforming an OO-Method object model into an UML class model, and vice versa. However, the OO-Method object model is only one of the diagrams involved in the definition of the OO-Method conceptual model, which is used by the OlivaNova technology to automatically generate software applications throw a MDD process. There are other diagrams involved in the definition of the OO-Method conceptual model, for instance, the *Presentation Model*, which allows the user interface of the generated applications to be defined.

The object model has been selected for the interchange between OO-Method and UML because it is the core diagram for the definition of the OO-Method conceptual model. In addition, due to its proximity with the UML class model, it could be more intuitive for defining the OO-Method object model using UML tools for those customers that already have experience in UML.

Figure 10 shows a UML model that has been defined using the UML profile generated to represent the concepts of the OO-Method association. This UML model describes the same example presented in Figure 8 (an association between the classes *Passenger* and *Flight*). Figure 10 also shows the description of the UML model in a tree view, where the application of the different stereotypes can be observed.

It is important to note that even though the OO-Method association is a binary association as the example presented in the UML profile generation process, the OO-Method association requires additional properties that are not defined in that simplified example.

The UML model presented in Figure 10 has been specified using the Eclipse UML2 tool [7]. This figure shows some of the OO-Method concepts that do not exist in UML, for instance, the concept of *shared event* that is represented by the stereotype *sharedEvent*. The shared events are services that are defined to manage the creation or destruction of links between instances of associated classes. Figure 10 also shows that the representation of the OO-Method class is performed by the stereotype *oOmClass*.

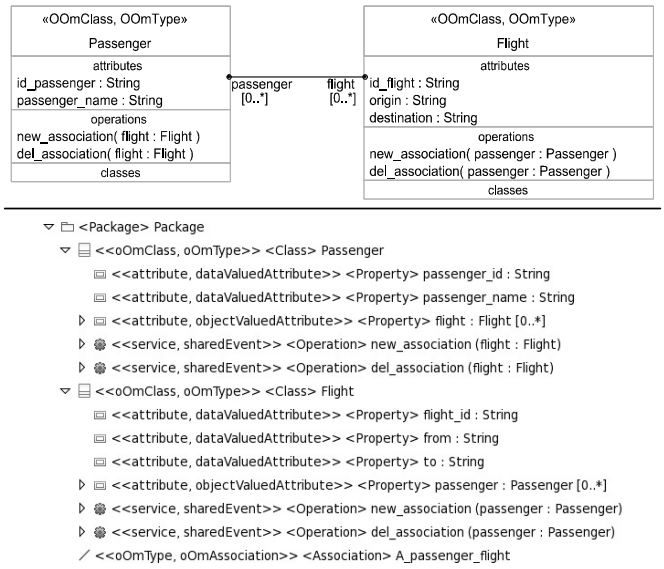


Figure 10. UML model extended with the OO-Method UML profile

Later, this UML model is transformed into an equivalent OO-Method object model by means of the XMI importer tool (see Figure 11).

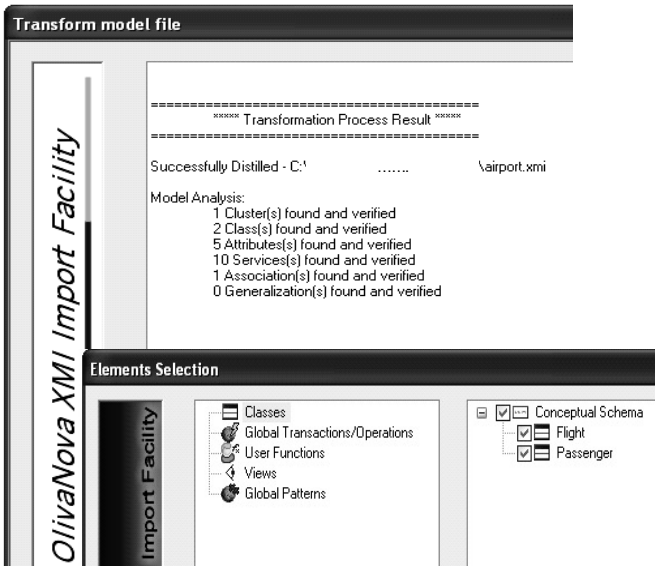


Figure 11. XMI importer application example

The XMI importer tool uses the mapping information obtained in the generation of the OO-Method profile (see Figure 9) to perform the transformations presented in the interchange proposal (see Figure 7).

The presentation model can be defined from the obtained OO-Method object model using the OO-Method presentation model editor, which is based on the OO-Method DSML. Figure 12 shows a screenshot of this tool, which presents a partial view of the presentation model related to the imported UML model. This figure shows that the executable application will have two Population Interaction Units (PIU) to represent the instances related to each class of the model.

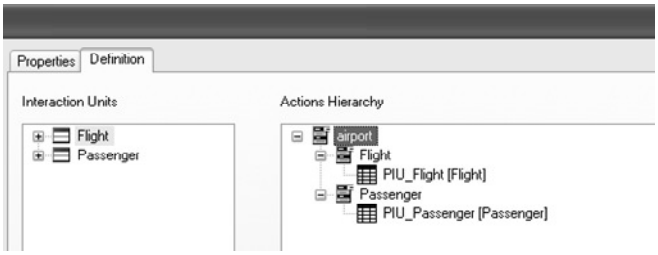


Figure 12. OO-Method presentation model

Finally, the resultant conceptual model is automatically implemented into an executable application by means of the OO-Method model compiler [24] (see Figure 13).

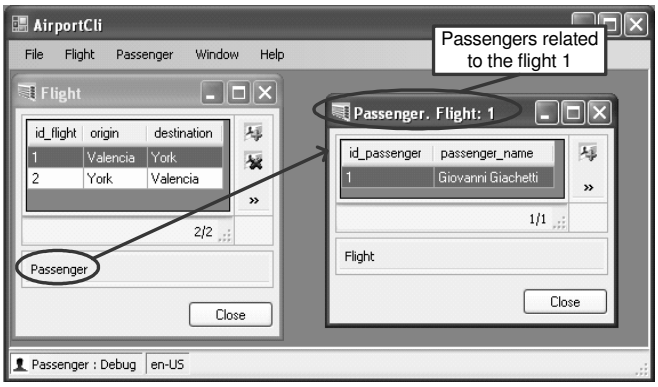
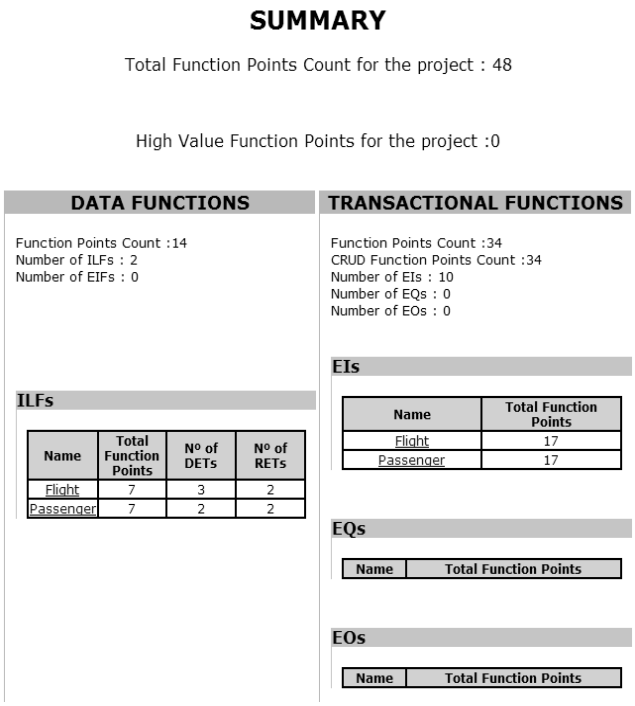


Figure 13. Generated application

Figure 13 shows a screenshot of the application that is generated from the imported UML class model and the defined OO-Method presentation model. This figure also shows that the association between the classes (defined in the UML model) is used to implement the navigation between related instances.

Finally, the imported UML model can also be used to obtain other software products; for instance, to obtain the functional size of the modeled application with the OO-Method function points tools [10][16] in order to estimate the cost of the generated application. Figure 14 shows a general view of the report of function points that has been automatically generated from the example UML model. This report presents

the functional size obtained by a measurement procedure defined according to the IFPUG FPA standard.



SUMMARY

Total Function Points Count for the project : 48

High Value Function Points for the project :0

DATA FUNCTIONS

Function Points Count :14
Number of ILFs : 2
Number of EIFs : 0

ILFs

Name	Total Function Points	Nº of DETs	Nº of RETs
Flight	7	3	2
Passenger	7	2	2

TRANSACTIONAL FUNCTIONS

Function Points Count :34
CRUD Function Points Count :34
Number of EIs : 10
Number of EQs : 0
Number of EOs : 0

EIs

Name	Total Function Points
Flight	17
Passenger	17

EQs

Name	Total Function Points

EOs

Name	Total Function Points

Figure 14. Functional size of the application generated for the example UML model

VI. RELATED WORK

To review the related works, we need to focus on those proposals oriented to the definition of UML profiles. In the literature, two main working schemas can be observed: 1) the definition of the UML profile from scratch; and 2) the definition of the UML profile starting from the *DSML Metamodel* [27], which is the metamodel that describes the conceptual constructs required by a MDD approach. For the interchange proposal presented in this paper, the second working schema has been selected since it provides a methodological solution that has more automation possibilities. Thus, the information that is required to perform the interchange of models is automatically obtained through the automation of the UML profile generation.

One of the first proposals related to the generation of a UML profile form a DSML metamodel is the work presented by Fuentes-Fernández et al. in [9], who propose some basic guidelines for the UML profile definition. In [27], Selic proposes a systematic approach that takes into account the new UML profile extension features. In addition, this systematic approach establishes some guidelines to ensure a correct DSML metamodel specification. It also defines some criteria to obtain a UML profile by means of a mapping that identifies the equivalences between the DSML metamodel and the UML metamodel. The definition of the mapping between the involve metamodels provides an initial idea about how the UML profile generation can be automated by means of models transformations that are guided by the participant metamodels. In this context, we can found the Lagarde et al. approach [15], which performs a partial automation through the identification of a set of specific design patterns. This approach requires a manual definition of an initial UML profile skeleton, which provides the information of the equivalencies between the DSML metamodel and the UML metamodel. This UML profile skeleton is used instead of the mapping between the involved metamodels. However, to generate the UML profile skeleton, it is necessary to know how the UML profiles are defined, which is opposite to the idea of the automatic UML profile generation that is oriented to encapsulate the complexity of the UML profile definition. In addition, since the patterns provided are only focused on a subset of the DSML metamodel constructs, the generation of the UML profile is not complete. Another interesting work is presented by Wimmer et al. [28]. This work proposes a semi-automatic approach that introduces a specific language to define the mapping between the DSML metamodel and the UML metamodel. This mapping allows an automated UML profile generation. However, this approach does not support all the possible mapping alternatives, for instance, the mapping M:M (many elements of the DSML metamodel mapped to many elements of the UML metamodel). As a consequence, the effective application in real MDD approaches is not possible.

In general, none of the works mentioned above provide a sound transformation process to automatically generate a complete UML profile solution. The main limitation of these approaches comes from the structural differences between the DSML metamodel and the UML metamodel such as the example presented in Figure 3. For this reason, we consider the generation of an Integration Metamodel [12] from the DSML metamodel in order to obtain an appropriate input for the automatic UML profile generation, which integrates the abstract syntax that is represented in a DSML metamodel into the UML metamodel. The transformation rules to perform this automatic generation are presented in [11].

Nevertheless, as this paper shows, the automatic UML profile generation can provide another interesting result, which is the mapping information that allows the interchange between the DSML and UML based models to be performed. In this context, proposals such as the Abouzahara et al. [1] state that it is possible to automate the interchange of UML models extended with a UML profile and DSML models by means of model transformations. However, this proposal requires the manual definition of a mapping between the DSML metamodel and the UML metamodel extended with the UML profile. Since this mapping is defined independently of the definition of the UML profile, there exists the risk that the obtained mapping does not reflect the equivalencies between the participant metamodels in a correct way. In our proposal this risk is avoided because the mapping is obtained during the UML profile generation. In addition, the complexity related to the correct design of a UML profile is encapsulated in the transformation of the Integration Metamodel [3]. Therefore, the mapping obtained from this transformation provides a correct identification of the equivalencies between the generated UML profile and the DSML metamodel (using the Integration Metamodel as intermediate model).

VII. CONCLUSIONS

In this paper, a proposal to obtain a hybrid modeling schema that integrates UML and DSMLs is presented. With this hybrid schema, the existent UML-based tools can be reused in the application of specific MDD solutions, thereby reducing the effort of implementing specific MDD tools. This interchange proposal is briefly exemplified by means of an implementation performed for the OO-Method industrial approach [3]. This example shows that it is possible to combine the use of UML and DSML tools for modeling the conceptual models that are required by MDD approaches.

Since the OO-Method implementation of this interchange proposal cannot be freely distributed, as further work we plan to finish the development of an open-source tool that implements this solution.

ACKNOWLEDGEMENTS

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

REFERENCES

[1] A. Abouzahra, J. Bézivin, M.D.D. Fabro, F. Jouault: "A Practical Approach to Bridging Domain Specific Languages with UML profiles". Best Practices for Model Driven Software Development (OOPSLA'05), 2005

[2] J. Bruck, K. Hussey: Customizing UML: Which Technique is Right for You? IBM, 2007

[3] CARE-Technologies: Web site, http://www.care-t.com/

[4] Eclipse: ATL Project, http://www.eclipse.org/m2m/atl/

[5] Eclipse: Eclipse Modeling Framework Project, http://www.eclipse.org/modeling/emf/

[6] Eclipse: Graphical Modeling Framework Project, http://www.eclipse.org/gmf/

[7] Eclipse: UML2 Project, http://www.eclipse.org/uml2/

[8] R.B. France, S. Ghosh, T. Dinh-Trong, A. Solberg: Model-driven development using uml 2.0: Promises and pitfalls. In: IEEE Computer, vol. 39 nº 2, 2006, pp. 59–66

[9] L. Fuentes-Fernández, A. Vallecillo: An Introduction to UML Profiles. In: The European Journal for the Informatics Professional (UPGRADE), vol. 5 nº 2, 2004, pp. 5–13

[10] G. Giachetti, B. Marín, N. Condori-Fernández, J.C. Molina: "Updating OO-Method Function Points". 6th IEEE International Conference on the Quality of Information and Communications Technology (QUATIC 2007), 2007, pp. 55–64.

[11] G. Giachetti, B. Marín, O. Pastor: "Using UML as a Domain-Specific Modeling Language: A Proposal for Automatic Generation of UML Profiles". 21st Conference on Advanced Information Systems Engineering (CAiSE'09). LNCS. Springer, 2009

[12] G. Giachetti, F. Valverde, O. Pastor: "Improving Automatic UML2 Profile Generation for MDA Industrial Development". 4th International Workshop on Foundations and Practices of UML (FP-UML) – ER Workshop. LNCS. Springer, 2008, pp. 113–122

[13] D. Harel, B. Rumpe: Meaningful Modeling: What's the Semantics of "Semantics"? In: IEEE Computer, vol. 37 nº 10, 2004, pp. 64–72

[14] B. Henderson-Sellers: "On the Challenges of Correctly Using Metamodels in Software Engineering". 6th Conference on Software Methodologies, Tools, and Techniques (SoMeT), 2007, pp. 3–35

[15] F. Lagarde, H. Espinoza, F. Terrier, S. Gérard: "Improving UML Profile Design Practices by Leveraging Conceptual Domain Models". 22th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2007, pp. 445–448

[16] B. Marín, G. Giachetti, O. Pastor: "Automating the Measurement of Functional Size of Conceptual Models in an MDA Environment". Product-Focused Software Process Improvement (PROFES). LNCS. Springer, 2008, pp. 215-229

[17] B. Marín, G. Giachetti, O. Pastor: "Intercambio de Modelos UML y OO-Method". X Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software (IDEAS), 2007, pp. 283–296

[18] OMG: Catalog of UML Profile Specifications

[19] OMG: MOF 2.0 Core Specification

[20] OMG: Object Management Group Web site, http://www.omg.org/

[21] OMG: UML 2.1.2 Infrastructure Specification

[22] OMG: UML 2.1.2 Superstructure Specification

[23] O. Pastor, J. Gómez, E. Insfrán, V. Pelechano: The OO-Method Approach for Information Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming. In: Information Systems. Elsevier Science, vol. 26 nº 7, 2001, pp. 507–534

[24] O. Pastor, J.C. Molina, E. Iborra: Automated production of fully functional applications with OlivaNova Model Execution. ERCIM News nº 57, 2004

[25] R. Pohjonen, S. Kelly: Domain-Specific Modeling. Dr. Dobb's Journal, 2002

[26] B. Selic: The Pragmatics of Model-Driven Development. In: IEEE Software, vol. 20 nº 5, 2003, pp. 19–25

[27] B. Selic: "A Systematic Approach to Domain-Specific Language Design Using UML". 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), 2007, pp. 2–9

[28] M. Wimmer, A. Schauerhuber, M. Strommer, W. Schwinger, G. Kappel: "A Semi-automatic Approach for Bridging DSLs with UML". 7th OOPSLA Workshop on Domain-Specific Modeling (DSM), 2007, pp. 97–104