# Using UML as a Domain-Specific Modeling Language: A Proposal for Automatic Generation of UML Profiles

Giovanni Giachetti, Beatriz Marín, and 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.** Nowadays, there are several MDD approaches that have defined Domain-Specific Modeling Languages (DSML) that are oriented to representing their particular semantics. However, since UML is the standard language for software modeling, many of these MDD approaches are trying to integrate their semantics into UML in order to use UML as DSML. The use of UML profiles is a recommended strategy to perform this integration allowing, among other benefits, the use of the existent UML modeling tools. However, in the literature related to UML profile construction; it is not possible to find a standardized UML profile generation process. Therefore, a process that integrates a DSML into UML through the automatic generation of a UML profile is presented in this paper. This process facilitates the correct use of UML in a MDD context and provides a solution to take advantage of the benefits of UML and DSMLs.

**Keywords:** UML Profile, UML, MDD, DSML.

## 1 Introduction

An appropriate modeling language is one of the most important elements for Model-Driven Development (MDD) approaches [22]. To obtain modeling languages that are adequate, different MDD approaches have defined their own Domain-Specific Modeling Languages (DSML) in order to represent their particular modeling needs. Two of the benefits that the use of DSMLs provide to MDD approaches are: (1) a correct and precise representation of the conceptual constructs related to the application domain, and (2) simplification of the implementation of tools oriented to improving the modeling tasks, development, and maintenance of generated software solutions.

However, since UML is seen as the standard language for software modeling purposes, many MDD approaches are integrating their modeling needs into UML in order to use UML as DSML. To perform this integration, the use of the extension mechanism defined in the UML specification, called *UML profile*, is the most suitable strategy. Therefore, the MDD approaches could achieve a larger market (greater number of potential users), take advantage of the existent UML technologies, and reduce the learning curve [2][12][23]. In addition, UML can be used as a mechanism to interchange ideas and theories among different research communities.

P. van Eck, J. Gordijn, and R. Wieringa (Eds.): CAiSE 2009, LNCS 5565, pp. 110–124, 2009.

Currently, there are many definitions of UML profiles that are associated to MDD approaches [14]. Generally speaking, these profile definitions are manually elaborated in a straightforward way and without a standardized process because a standard that specifies how the UML extensions must be defined does not currently exist [6]. For this reason, many of the existent UML profiles are invalid or of poor quality [23]. In addition, the manual definition of a UML profile is an error-prone and time-consuming task [24]. These two risk factors (time and error) must be avoided, especially in a MDD industrial context, where time costs money and mistakes in implementation directly impact on customer satisfaction.

To avoid the risks described above, some works related to UML profile elaboration have defined proposals to achieve a semi-automated profile generation [12][24]. For the generation of the UML profile, these proposals use as input the metamodel that describes the *conceptual constructs* related to the DSML of an MDD approach (the *DSML Metamodel*). However, none of these proposals provide a sound solution for the automatic generation of a complete UML profile. This is because, in real MDD solutions, structural differences between the DSML metamodel and the UML Metamodel, which prevent the automated identification of the extensions that must be performed in UML, may be found.

This paper introduces a solution for a completely automated UML profile generation using as input the DSML Metamodel related to a MDD approach. This solution is part of an integration process that has been designed to introduce the modeling needs of MDD approaches into UML. In this process, the proposal presented in [8] is used to obtain a correct input for the generation of the UML profile.

Thus, this paper shows how the required UML extensions can be automatically identified and details the transformation rules to obtain the UML Profile that implements these extensions. This paper also shows the application of the integration process in an industrial MDD approach called *OO-Method* [19][20] in order to exemplify how this process can be used to integrate UML and DSML models in a unique MDD solution.

The rest of the paper is organized as follows: Section 2 shows the background related to UML profile generation. Section 3 introduces the proposed process. Section 4 details the automatic UML profile generation. Section 5 presents the application of the process. Finally, Section 6 presents our conclusions and further work.

## 2 Background

The UML profile extension mechanism is defined in the UML Infrastructure [16]. It defines the mechanisms used to adapt existing metamodels to specific platforms, domains, business objects, or software process modeling. In this work, the UML profiles are used to integrate the modeling needs of MDD approaches in UML [17]. Further details about UML Profiles and UML extensions can be consulted in [2][7][16].

In the literature related to the definition of UML profiles, 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* [23], which is the metamodel that describes the conceptual constructs required by a MDD approach. For

the process presented in this paper, the second working schema has been selected since it provides a methodological solution that has more automation possibilities.

One of the first proposals related to this working schema is the work presented by Fuentes-Fernández et al. in [7], who propose some basic guidelines for the UML profile definition. In [23], 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 and 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.

Nowadays, there are very few works related to the automation of the definition of a UML profile. One of these works is the Lagarde et al. approach [12], which can be partially automated through the identification of a set of specific design patterns. However, this approach requires the manual definition of an initial UML profile skeleton. Another interesting work is presented by Wimmer et al. [24]. 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, the analyzed works are only centered on representing those modeling elements of the DSML that do not exist in UML using the generated UML Profile. However, this focus is not enough to generate a correct UML profile because there are other elements that must be considered for a correct UML profile definition. These other elements are: 1) the representation of the differences that exist between elements of the DSML, and corresponding elements that already exist in UML, and 2) the definition of rules oriented to validate the correct use of the UML profile in order to produce correct conceptual models.

Even when these additional considerations are omitted, 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. If these structural differences are solved, then the UML Profile generation can be automated. In Giachetti et al.[10], we propose a solution to solve these structural problems. This solution consists of the transformation of the DSML metamodel into a new metamodel. This new metamodel provides an adequate input to automate the integration of the abstract syntax that is represented in a DSML metamodel into the UML Metamodel. The automatic UML profile generation that is presented in the next section of this paper is based on this solution.

## 3 A Process to Integrate a DSML into UML

This section presents the process that has been defined to integrate the modeling needs related to a specific MDD approach into UML by means of an automatically generated UML profile. These modeling needs are represented by the DSML

metamodel of the MDD approach. To elaborate this process, different works have been considered. Some of these works are: definition of profiles using DSML metamodels [7][12][23][24], correct use of metamodels in software engineering [11], UML profile implementations [14], interchange between UML and DSMLs [1][9], and new UML profile features introduced in UML [16].

The proposed process can be used in those MDD approaches where the conceptual constructs can be considered as a subset or extension of the UML constructs. This constraint comes from the limitation of the UML profile to change the reference metamodel and guarantees that the MDD approaches that use the proposed process are UML-Compliant. The process is composed of three steps (see Figure 1):

- *Step 1:* Definition of the Integration Metamodel from the DSML metamodel taking into account the UML Metamodel defined in the UML Superstructure [17].
- *Step 2:* Comparison between the Integration Metamodel and the UML Metamodel. This comparison identifies the extensions that must be defined in UML by using the equivalences identified in the *Step 1*.
- *Step 3:* Transformation of the Integration Metamodel according to a set of transformation rules in order to obtain the final UML profile.

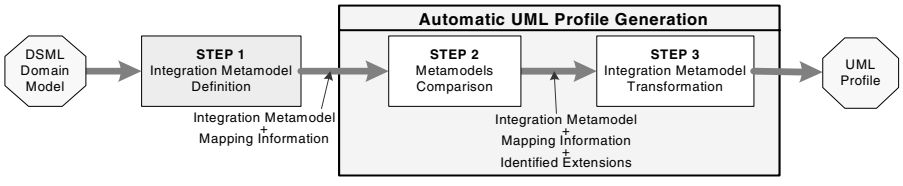


**Fig. 1.** Integration Process Schema

The second and third steps of the integration process correspond to the *Automatic UML Profile Generation*. In this process, the original DSML metamodel is redefined to obtain the input required for the UML profile generation. This input is called the *Integration Metamodel*, and its main characteristics are described below.

### 3.1 Definition of the Integration Metamodel

The Integration Metamodel is a special DSML metamodel that has been defined to automate the integration of a DSML 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. This mapping information is included inside the Integration Metamodel definition. The UML metamodel selected for the definition of the Integration Metamodel is the metamodel presented in the UML Superstructure [17].

The Integration Metamodel is manually defined according to the systematic approach presented in [10]. In that work, the structural problems that can exist in a

DSML metamodel as well as the benefits of the Integration Metamodel are discussed. Summarizing, 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 [15]. By using EMOF, the resultant metamodel properties do not have features that are not supported by UML profiles. Moreover, EMOF has a standardized XMI definition [18]. Thus, the UML profile generation can be automated by means of transformation rules that are implemented over the XMI definition of the Integration Metamodel. The EMOF definition also allows the implementation of specific model editors with tools such as Eclipse GMF [4].
- It is mapped to the UML Metamodel taking into account: Classes, Properties (Attributes and Associations), Enumerations, Enumeration Literals, and Data Types. The mapped elements are considered as *equivalent elements*, and the non-mapped elements are considered as *new elements* in the Integration Metamodel. This mapping complies with the following rules:
  - All the classes from the Integration Metamodel are mapped. This assures that the conceptual constructs of the DSML can be represented from the conceptual constructs of UML.
  - The mapping is defined between elements of the same type (classes with classes, attributes with attributes, and so on).
  - An element from the Integration Metamodel is only mapped to one element of the UML Metamodel. This rule also considers the possibility of have X:1 mappings ($X \geq 0$); for instance, many classes of the Integration Metamodel can be mapped to one class of UML. In this example, the mapping rule is also accomplished because each class of the Integration Metamodel is only mapped to one UML class. It is important to note that the many-to-many mappings that may exist between the original DSML metamodel and the UML Metamodel are transformed into X:1 mappings during the generation of the Integration Metamodel.
  - If the properties (attributes and associations) of a class from the Integration Metamodel are mapped to properties of a UML class, then the class that owns the properties is mapped to this UML class (or a generalization of it).

## 4 Automatic Generation of the UML Profile

This section presents how a correct UML profile can be automatically generated from an Integration Metamodel. This automatic generation is comprised by two steps: 1) the comparison of metamodels to obtain the required UML extensions; and 2) The transformation of the Integration Metamodel into the corresponding UML profile. These steps are presented below.

### 4.1 Comparison of Metamodels

The identification of the required UML extensions is performed 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 comparison between the Integration Metamodel and the UML Metamodel considers:

- The identification of *new elements*, which are the elements from the Integration Metamodel that are not equivalent 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).

Figure 2 shows an example of an Integration Metamodel that will be used to help understand how the UML extensions are identified. The metamodel presented in this figure represents a binary association between classes. In this metamodel, the attributes of the class *DMAssociationEnd* represent the cardinality related to each association end, and the attributes of the classes *DMClass* and *DMAssocation* represent generic attributes related to these classes.

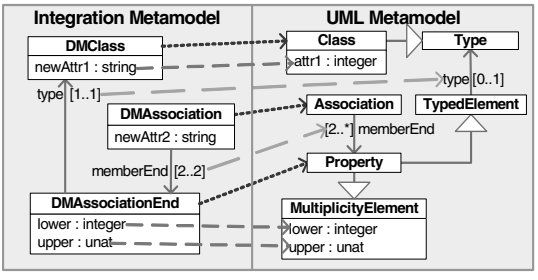


**Fig. 2.** Integration Metamodel related to a binary association between classes

Table 1 shows the comparison result obtained from the Integration Metamodel presented in Figure 2. In this table, the column *Integration Metamodel* shows the *new element*s identified, or *equivalent elements* that differ in relation to the related UML elements. 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 1.** Metamodel comparison for the Integration Metamodel presented in Figure 2

| **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<br>Different type: I.M. = DMClass; UML = Type |

The mapping information defined in the Integration Metamodel allows the identification of type differences. For instance, in the case of *DMAssociationEnd.type* and *Property.type*, the type is different because *DMClass* is not equivalent to *Type*.

The cardinality differences are identified by analyzing the lower and upper bound of the equivalent properties and the referenced UML properties. This is the case of the equivalent properties *DMAssociation.memberEnd* and *DMAssociationEnd.type*.

Finally, the differences identified in the comparison are the extensions that must be introduced into the UML in order to correctly represent the modeling needs of the related MDD approach.

### 4.2 Transformation of the Integration Metamodel

The third step in the process defines a set of transformation rules to automatically generate a complete UML profile from the Integration Metamodel and the UML extensions previously obtained. These transformation rules are defined considering that the *new elements* and the differences between *equivalent elements* identified during the metamodel comparison must be represented in the generated UML profile. In addition, these rules take into account the automatic generation of the needed constraints in order to assure the correct application of the generated extensions.

In order to show how the Integration Metamodel can be transformed into the corresponding UML profile, the required transformation rules are described below. These transformation rules are separated by the different EMOF conceptual constructs. The possible modeling situations are analyzed for each construct, according to the Integration Metamodel features. A figure that exemplifies the application of the transformation rules in a generic way is also presented.

**Classes**

Rule 1: One Stereotype for each *equivalent class*. The stereotype extends the referenced UML class, and its name is equal to the *equivalent class* name. Figure 1 exemplifies this rule.

This first transformation rule is the most relevant because it involves the generation of the stereotypes, which are the main constructs of the UML profile. The rest of transformation rules are applied according to the results obtained by this first rule.

Validation: At the end of the UML profile generation, if there is only one stereotype that extends a UML class, then the stereotype extension must be defined as *required*. This constraint is defined because, in the DSML context, the UML class only has the semantics of the involved equivalent class (see *Class3* in Figure 3).

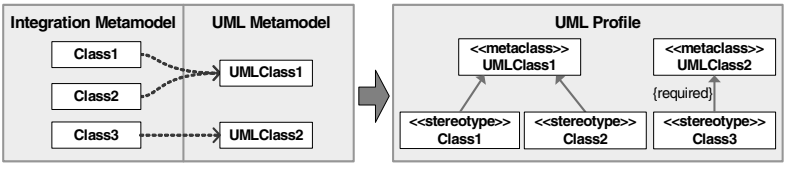


**Fig. 3.** Generic example for the transformation rule related to classes: Rule 1

**Properties**

In EMOF, the properties represent attributes of a class (metaclass) or references (associations) between the classes. The main difference between an attribute and an association is that an attribute represents a data-valued property, while an association is an object-valued property. In other words, in an association, the type is given by another class of the model that represents the related class. These differences are taken into account in the definition of the involved transformation rules.

Rule 2: One tagged value for each *new property*. The tagged value must have the same type and cardinality as the *new property*. The name of the tagged value must be the name of the *new property*. In the case of an association, the tagged value must have the same aggregation kind as the *new property*. The application of this rule can be observed in Figure 4 for the association *Class1.rolClass2*.

Rule 3: One OCL constraint if the lower bound of an *equivalent property* is higher than the lower bound of the referenced UML property:

```
self.[property]->size() >= [newLowerBound]
```

Rule 4: One OCL constraint if the upper bound of an *equivalent property* is lower than the upper bound of the referenced UML property:

```
self.[property]->size() <= [newUpperBound]
```

As Figure 4 shows, rules 3 and 4 are applied to the *Class2.roleClass3* and *Class3.roleClass1*, respectively.

Validation: For rules 3 and 4, an OCL constraint is defined to validate that the corresponding stereotype is applied each time that the involved UML association is established. Thus, the type of the referenced UML association is restricted to the stereotype that represents the type of the *equivalent association*:

```
self.[equivalentAssociation]->isStereotyped[1]([newType])
```

This validation is also applied if the type of an *equivalent association* is changed by a specialization of the original type (see *Class2.rolClass3* in Figure 4).

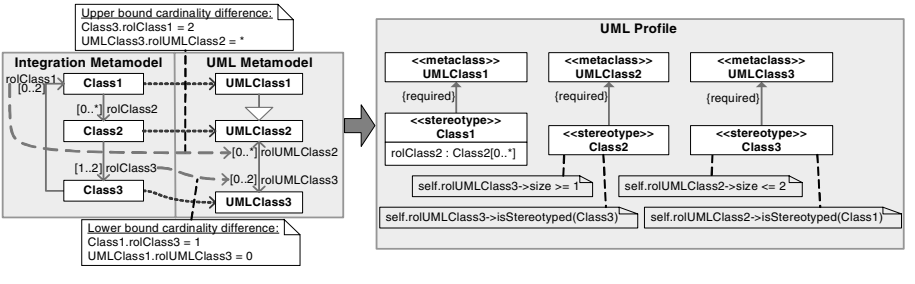


**Fig. 4.** Generic example for the transformation rules 2 to 4

[1] The OCL operation *isStereotyped* is not part of the OMG specification and is only used to simplify the OCL rules presented. In the application of the integration process, this operation must be implemented according to the target UML tool.

Even though an extension relationship represents a refinement of a class in a way similar to a generalization relationship, its semantics is represented as a special kind of association and not as a generalization. For this reason, a tagged value cannot redefine UML properties. Therefore, when the differences that exist between an *equivalent property* and the referenced UML property cannot be represented using OCL constraints, a tagged value that replaces the original UML property is created. In this case, the MDD process must only consider the tagged value and not the UML property.

Rule 5: One tagged value that replaces a UML property when one of the following conditions holds:

- The type of *equivalent property* is different than the type of the referenced UML property, and the new type is not a specialization of the original type or a stereotype that extends the original type (see *Class1.attr3* in Figure 5).
- The upper bound of the *equivalent property* is higher than the upper bound of the referenced UML property (see *Class1.attr2* in Figure 5).
- The lower bound of the *equivalent property* is lower than the lower bound of the referenced UML property (see *Class1.attr1* in Figure 5).

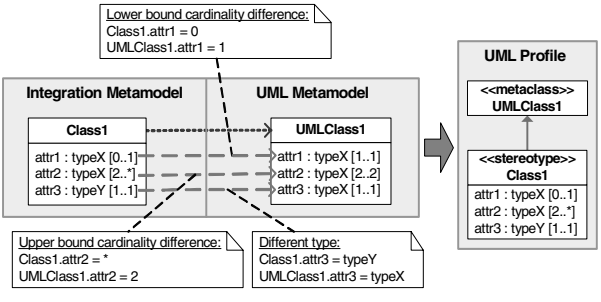


**Fig. 5.** Generic example for transformation rule 5

**Enumerations**

The enumerations are used to specify a customized set of values that can be represented by an attribute of a class. Graphically, the enumerations are represented as a class. However, the enumeration is a specialization of a *Classifier* and not of a *Class*. This difference is considered in the following transformation rule.

Rule 6: One enumeration for each *new enumeration* or *equivalent enumeration* with new literal values. In the case of an *equivalent enumeration*, the generated enumeration replaces the original UML enumeration, and the involved *equivalent attributes* are considered as *new attributes* (Rule 2). In this case, since the UML enumeration is not a class, it cannot be extended with a stereotype in order to include the new literal values. Figure 6 shows the application of this rule for *Enum2* (*equivalent enumeration*) and *Enum3* (*new enumeration*).

Validation: One OCL constraint for each attribute whose type corresponds to an *equivalent enumeration* that has fewer alternatives (literal values) than the referenced UML enumeration (see *Class1.attr1* and *Enum1* in Figure 6).

```
self.[attribute] <> #[nonMappedLiteralValue]
```

This constraint avoids the use of invalid alternatives (non-referenced literal values) that are defined in the referenced UML enumeration.

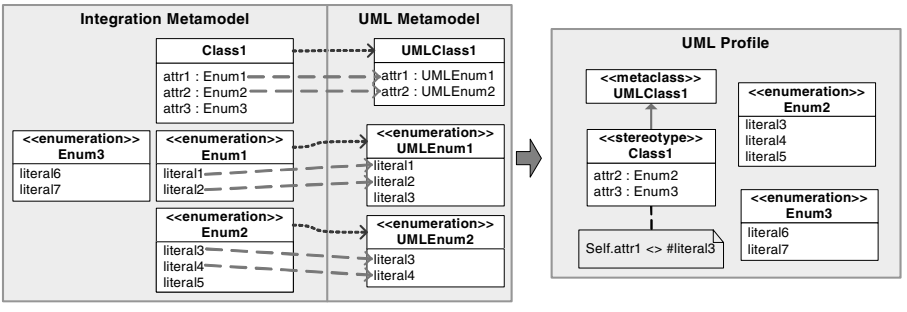


**Fig. 6.** Generic example for transformation rule 6.

**Generalizations**

The generalization relationships have interesting features that must be considered in the generation of the related stereotypes. Two of the main features that must be considered are: 1) since stereotypes are a special kind of class, it is possible to define a generalization between stereotypes; and 2) since the extension association between a stereotype and its related class is a specialization of *Association*, the extension relationship can be inherited.

Rule 7: Define one generalization between two stereotypes that represent *equivalent classes* that are associated with a *new generalization* and that are referencing the same UML class. The extension related to the child stereotype is not defined since it is implicit in the generalization relationship. Figure 7 shows the application of this rule for the generalization defined between the classes *Class1* and *Class3*.

Rule 8: If there is a *new generalization* between two *equivalent classes* that are referencing different UML classes, the generalization relationship is not represented in the UML profile. In this case, the extensions of each stereotype to the corresponding UML class are defined, and the inherited properties (attributes and associations) are duplicated (see the generalization between classes *Class3* and *Class4* in Figure 7). If the generalization is represented, then the child stereotype will be able to extend the UML class that is extended by the father stereotype. This could produce a modeling error since, according to the mapping information, the child stereotype is referencing (extends) a different UML class.

Rule 9: If there is an *equivalent generalization* between two *equivalent classes*, the generalization relationship is not represented in the UML profile, and only the

extensions of each stereotype are defined to the corresponding UML class. In this way, the generalization defined in UML is used instead of the *equivalent generalization* (see the generalization between classes *Class1* and *Class2* in Figure 7).

Note that an *equivalent generalization* represents a generalization that already exists in the UML Metamodel. The *equivalent generalizations* are automatically identified through the participant classes of the Integration metamodel that are equivalent to the classes that participate in the UML generalization.

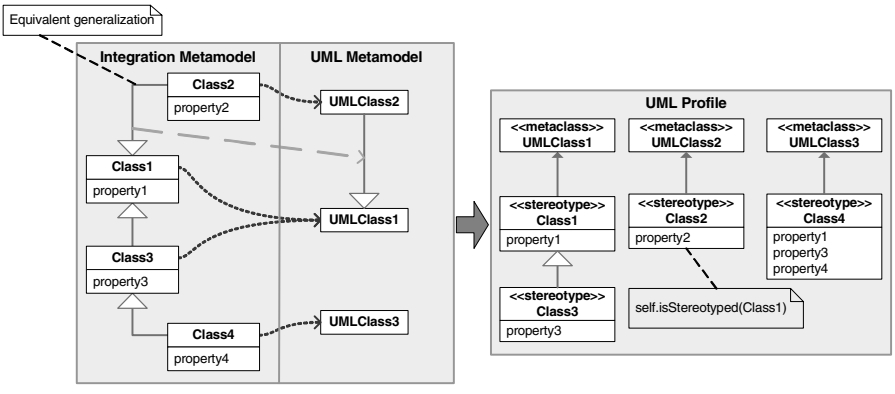


**Fig. 7.** Generic example for transformation rules 7, 8 and 9

**OCL Rules**

The OCL rules defined in the Integration Metamodel manage the interactions between the different constructs of the DSML. Therefore, these rules must be included in the generated UML profile.

Rule 10: The OCL rules defined in the classes of the Integration Metamodel must be included in the stereotypes generated from these classes. The elements referenced in the rules must be changed by the corresponding UML classes and stereotypes.

**Data Types**

Rule 11: The *new data types* defined in the Integration Metamodel are defined in a separate model library. This model library must be imported in each UML model that is designed using the generated UML profile.

The equivalent data types that have differences in relation to the referenced UML data types are considered as *new data types*. Since the data types are classifiers, they cannot be extended using stereotypes.

Validation: The UML data types that are not referenced by *equivalent data types* are not valid in the DSML context. For this reason, one OCL constraint is defined over the UML metaclass *TypedElement* to restrict the invalid UML data types:

```
self.type->oclIsTypeOf([invalidType]) = False
```

Rule 11 is the last rule defined for the transformation of the Integration Metamodel in the equivalent UML profile. Figure 8 presents the UML profile obtained after applying the proposed transformation rules to the example Integration Metamodel presented in Figure 2.

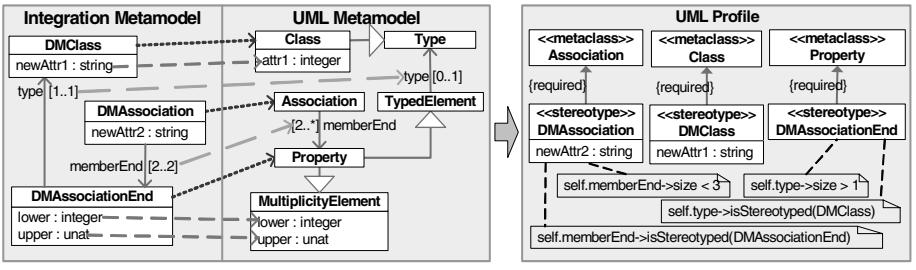


**Fig. 8.** UML profile generated for the example Integration Metamodel

In addition to the UML profile, the transformation of the Integration Metamodel also generates a new mapping that takes into account the generated UML profile elements (stereotypes, tagged values, etc.). This new mapping provides bidirectional equivalence between the Integration Metamodel and the UML Metamodel (extended with the generated UML profile). Figure 9 shows this new mapping information for the UML profile presented in Figure 8.

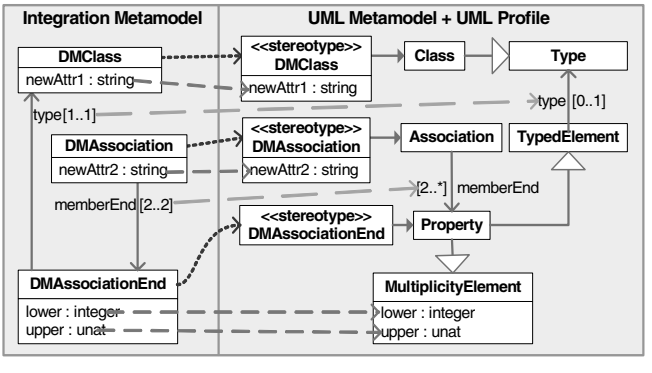


**Fig. 9.** New mapping information generated for the UML profile presented in Figure 8

The generated UML profile together with the new mapping definition can be used to interchange UML models and DSML models [9], in order to take advantage of these two modeling solutions. The following section shows how the proposed integration process has been applied to an industrial MDD approach [3], to integrate UML tools and the existent DSML-based tools.

## 5 Applying the Integration Process

In this section, the implementation schema used to apply the proposed integration process in the OO-Method industrial approach [19][20] is introduced. This schema (Figure 10) takes advantage of UML tools, without losing the benefits of the existent MDD technology based on the OO-Method DSML.

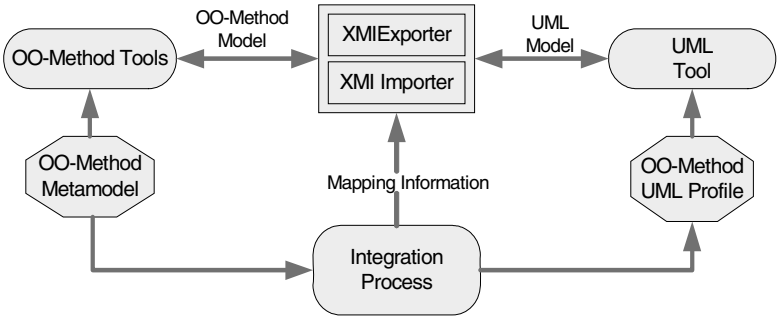


**Fig. 10.** Schema designed to apply the integration process into the OO-Method approach

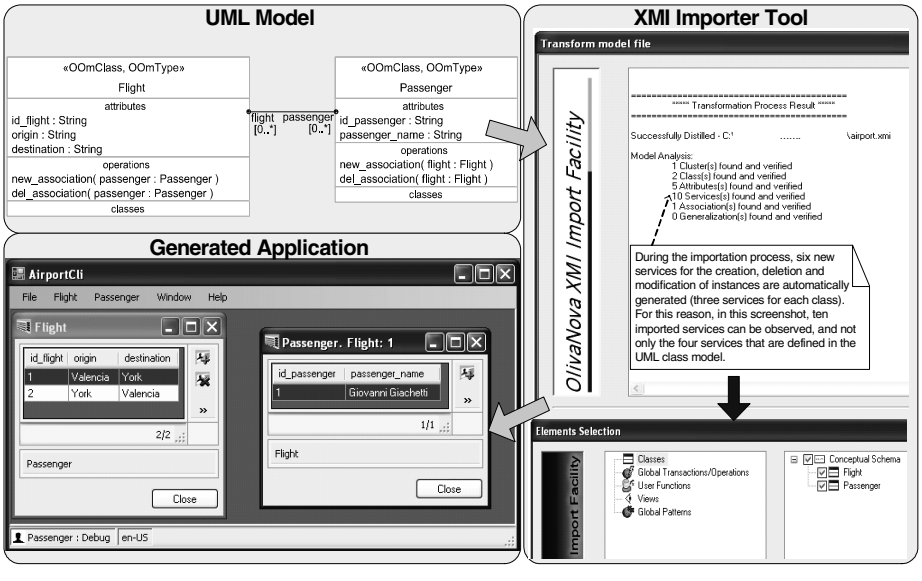


**Fig. 11.** Application of the OO-method compilation technology over a UML model

The core of the proposed schema is made up of two interchange tools called XMI importer and XMI exporter [3], which transform the UML models into DSML models, and vice versa. These tools are extended with the new mapping information obtained in the UML profile generation process in order to support the generated UML profile [9]. Figure 11 shows the application of the extended XMI importer tool to

automatically generate an application from a UML model extended with the OO-Method UML profile. This model has been defined using the Eclipse UML2 tool [5].

The schema proposed to apply the integration process in the OO-Method approach has three main benefits:

1. The technology, support, and commercial structure defined for the OO-Method development process can be used in a transparent way for UML users.
2. The different OO-Method tools, such as the OO-Method model compiler [21], and functional size measurement tools [8][13] can be used over UML models.
3. The customers can easily migrate from UML tools to OO-Method tools in order to take advantage of the improved functionalities that the OO-method tools provide for the management of OO-Method conceptual models.

## 6 Conclusions and Further Work

This paper presents a solution for the automatic generation of a UML profile from the metamodel that represents the DSML related to a MDD approach. This automatic generation is applied in an integration process in order to take advantage of the benefits provided by the use of UML and DSML based technologies.

The proposed solution tackles an important topic that has not yet received the required attention: the correct definition of UML profiles for MDD solutions. Even though the number of UML profile solutions has increased, the number of publications related to a correct UML profile definition is very limited [23]. In order to obtain this correct definition, the proposed transformations are focused on three main elements: 1) the generation of modeling elements defined in the DSML that are not present in UML; 2) the management of differences that could exist between elements of the DSML that are equivalent with UML elements; and 3) the generation of constraints to assure that the application of the generated UML profile follows the DSML specification.

It is important to note that the transformation rules defined in this paper are just one possible solution for the complete generation of a correct UML profile. Variations of these transformation rules can be defined depending on different design decisions. This paper also explains how this solution can be applied in MDD approaches, taking as example the application performed for the OO-Method approach in order to integrate UML tools and the existent OO-Method tools.

As further work, we plan to finish the implementation of a set of open-source tools that support the proposed integration process in order to provide a generic integration solution for different MDD approaches.

**Acknowledgments.** This work has been developed with the support of MEC under the project SESAMO TIN2007-62894 and co financed by FEDER.

## References

1. Abouzahra, A., Bézivin, J., Fabro, M.D.D., Jouault, F.: A Practical Approach to Bridging Domain Specific Languages with UML profiles. In: Best Practices for Model Driven Software Development (OOPSLA 2005) (2005)
2. Bruck, J., Hussey, K.: Customizing UML: Which Technique is Right for You? IBM (2007)

3. CARE-Technologies, http://www.care-t.com/
4. Eclipse: Graphical Modeling Framework Project, http://www.eclipse.org/gmf/
5. Eclipse: UML2 Project, http://www.eclipse.org/uml2/
6. France, R.B., Ghosh, S., Dinh-Trong, T., Solberg, A.: Model-driven development using uml 2.0: Promises and pitfalls. IEEE Computer 39(2), 59–66 (2006)
7. Fuentes-Fernández, L., Vallecillo, A.: An Introduction to UML Profiles. The European Journal for the Informatics Professional (UPGRADE) 5(2), 5–13 (2004)
8. Giachetti, G., Marín, B., Condori-Fernández, N., Molina, J.C.: Updating OO-Method Function Points. In: 6th IEEE International Conference on the Quality of Information and Communications Technology (QUATIC 2007), pp. 55–64 (2007)
9. Giachetti, G., Marín, B., Pastor, O.: Using UML Profiles to Interchange DSML and UML Models. In: Third International Conference on Research Challenges in Information Science, RCIS (2009)
10. Giachetti, G., Valverde, F., Pastor, O.: Improving Automatic UML2 Profile Generation for MDA Industrial Development. In: Song, I.-Y., et al. (eds.) ER Workshops 2008. LNCS, vol. 5232, pp. 113–122. Springer, Heidelberg (2008)
11. Henderson-Sellers, B.: On the Challenges of Correctly Using Metamodels in Software Engineering. In: 6th Conference on Software Methodologies, Tools, and Techniques (SoMeT), pp. 3–35 (2007)
12. Lagarde, F., Espinoza, H., Terrier, F., Gérard, S.: Improving UML Profile Design Practices by Leveraging Conceptual Domain Models. In: 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 445–448 (2007)
13. Marín, B., Giachetti, G., Pastor, O.: Automating the Measurement of Functional Size of Conceptual Models in an MDA Environment. In: Jedlitschka, A., Salo, O. (eds.) PROFES 2008. LNCS, vol. 5089, pp. 215–229. Springer, Heidelberg (2008)
14. OMG: Catalog of UML Profile Specifications
15. OMG: MOF 2.0 Core Specification
16. OMG: UML 2.1.2 Infrastructure Specification
17. OMG: UML 2.1.2 Superstructure Specification
18. OMG: XMI 2.1.1 Specification
19. Pastor, O., Gómez, J., Insfrán, E., Pelechano, V.: The OO-Method Approach for Information Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming. Information Systems 26(7), 507–534 (2001)
20. Pastor, O., Molina, J.C.: Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling, 1st edn. Springer, New York (2007)
21. Pastor, O., Molina, J.C., Iborra, E.: Automated production of fully functional applications with OlivaNova Model Execution. ERCIM News 57 (2004)
22. Selic, B.: The Pragmatics of Model-Driven Development. IEEE Software 20(5), 19–25 (2003)
23. Selic, B.: A Systematic Approach to Domain-Specific Language Design Using UML. In: 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), pp. 2–9 (2007)
24. Wimmer, M., Schauerhuber, A., Strommer, M., Schwinger, W., Kappel, G.: A Semi-automatic Approach for Bridging DSLs with UML. In: 7th OOPSLA Workshop on Domain-Specific Modeling (DSM), pp. 97–104 (2007)