A Rigorous Framework for Model-Driven Development

Liliana Favre

Universidad Nacional del Centro de la Provincia de Buenos Aires
Comisión de Investigaciones Científicas de la Provincia de Buenos Aires
Argentina

lfavre@exa.unicen.edu.ar

Abstract. The Model Driven Architecture (MDA) is an initiative of the Object Management Group (OMG) to model-centric software development. MDA distinguishes different kinds of models: Platform Independent Models (PIM), Platform Specific Models (PSM) and code models. Metamodeling plays a key role in MDA. A combination of formal specification techniques and metamodeling can help us to address Model-Driven Developments (MDD). In this paper we describe a MDA framework that comprises the NEREUS metamodeling notation, a system of transformation rules to bridge the gap between UML/OCL and NEREUS and, the definition of MDA-based components and model/metamodeling transformations. NEREUS can be viewed as an intermediate notation open to many other formal languages. In particular, we show how to integrate NEREUS with algebraic languages such as CASL.

1 Introduction

The Object Management Group (OMG) is facing a paradigm shift from object-oriented software development to model-centric development [18]. A recent OMG initiative is the Model Driven Architecture (MDA), which is emerging as a technical framework to improve productivity, portability, interoperability, and maintenance [16].

MDA promotes the creation of abstract models that are developed independently of a particular implementation technology and automatically transformed by tools into models for specific technologies. All artifacts such as requirements specification, architecture descriptions, design descriptions and code, are regarded as models. MDA distinguishes platform independent models (PIMs), platform specific models (PSMs) and code models. UML combined with the Object Constraint Language (OCL) is the most widely used way for writing either PIMs or PSMs [17].

In MDA, one of the key features is the notion of automatic transformations that are used to convert a model in a source language into a model in a target language. A Model-Driven Development (MDD) is carried out as a sequence of model transformations.

In this paper, we analyze the integration of MDD with knowledge developed by the formal method community. MDD can take advantage of the different formal languages and the diversity of tools developed for prototyping, model validations and model simulations.

We describe a MDA framework that facilitates model-to-model transformations, focusing on UML metamodels. It was designed to automate different tasks that are at the core of MDA, such as refining and refactoring.

The framework comprises a megamodel for defining MDA components, the metamodeling notation called NEREUS and, the definition of metamodeling/model transformations using UML/OCL and NEREUS.

A megamodel is a set of elements that represent and/or refer to UML-based models and metamodels at different levels of abstraction (PIMs, PSMs, and code) and structured by different transformation relationships. NEREUS can be viewed as an intermediate notation open to many other formal specifications such as algebraic, functional or logic ones. We define a system of transformation rules to transform automatically UML/OCL models into NEREUS. We also show how to convert NEREUS specifications to algebraic languages such as CASL [5].

This paper is organized as follows. Section 2 presents a megamodel for defining MDA-based components. Section 3 describes how to formalize metamodels in the intermediate notation NEREUS. Section 4 shows how to bridge the gap between UML/OCL and NEREUS. Section 5 describes how to integrate the intermediate notation NEREUS with the algebraic language CASL. Section 6 presents related work. Finally, Section 7 concludes and discusses further work.

2 A Megamodel for Model-Driven Development

Reusable components that will be used in a MDA-based process have to be described at different abstraction levels. To define reusable components we propose a megamodel that integrates PIMs, PSMs and code with their respective metamodels.

We define MDA components at three different levels of abstraction: Platform Independent Component Model (PICM), Platform Specific Component Model (PSCM) and Implementation Component Model (ICM). The PICM includes a UML/OCL metamodel that describes a family of all those PIMs that are instances of the metamodel. A PIM is a model that contains no information of the platform that is used to realize it.

A PICM-metamodel is related to more than one PSCM-metamodel, each one suited for different platforms. The PSCM metamodels are specializations of the PICM-metamodel. The PSCM includes UML/OCL metamodels that are linked to specific platforms and a family of PSM-models that are instances of the respective

PSCM-metamodel. Every one of them describes a family of instances (PSM-models). PSCM-metamodels correspond to ICM-metamodels (Fig.1).

Metamodels are expressed in a combination of UML class diagrams and OCL. The 4 main core metamodeling constructs are classes, binary associations, data types and package. A metamodel is a description of all the concepts that can be used in this level. For instance, the concepts of attribute, operations and associations are part of the PIM metamodel, the concepts of table, column and foreign keys are part of PSM-relational metamodel and the concepts of constructor and method are part of the Java metamodel.

A metamodeling transformation specifies a mapping between models built using types specified in metamodels. In this case, the transformation is a specification of a mechanism to convert the elements of a model conforming to a particular metamodel into elements of another model that conforms to another (possibly the same) metamodel.

Fig. 1 shows the different correspondences that may be held between several models and metamodels. For instance we use the megamodel to create specifications of solutions for several design patterns. The transformation of UML models can be supported by using design patterns that are described at three different abstraction levels: the PICM includes a PIM metamodel whose instances are several pattern solutions; the PSCM includes metamodels for particular platforms and, the ICM level includes metamodels for different programming languages. Metamodels avoid defining as many components as different pattern solutions can appear.

A transformation rule in OCL is defined by its name, a source model element, a target model element, a source condition, a target condition and a set of mapping rules. The source condition is an invariant that states the conditions that must be held in the source model for the transformation rule to be applied. The target condition is an invariant that must be held in the target model. A transformation between models is a sequence of OCL transformations. The transformation rules are declarative and have several features: bidirectionality, traceability and incremental consistency.

Developing reusable components requires a high focus on software quality. MDA can take advantages of formal languages and tools developed around them. The traditional techniques for verification and validation are still essential to achieve software quality. The formal specifications are of particular importance for supporting testing of applications, for reasoning about correctness and robustness of models and for generating code “automatically” from abstract models.

In this direction we investigate how to formalize MDA megamodels. The formalization implies to specify UML/OCL model/metamodels and model/metamodel transformations. Considering that different tools could be used to validate or verify models at different abstraction levels (PIMs, PSMs, or implementations), we propose to formalize the megamodel in an intermediate notation called NEREUS [12]. We define one bridge between UML/OCL and NEREUS. For a subsequent translation into formal languages, NEREUS may serve as a source language. In the following section we describe the NEREUS notation.

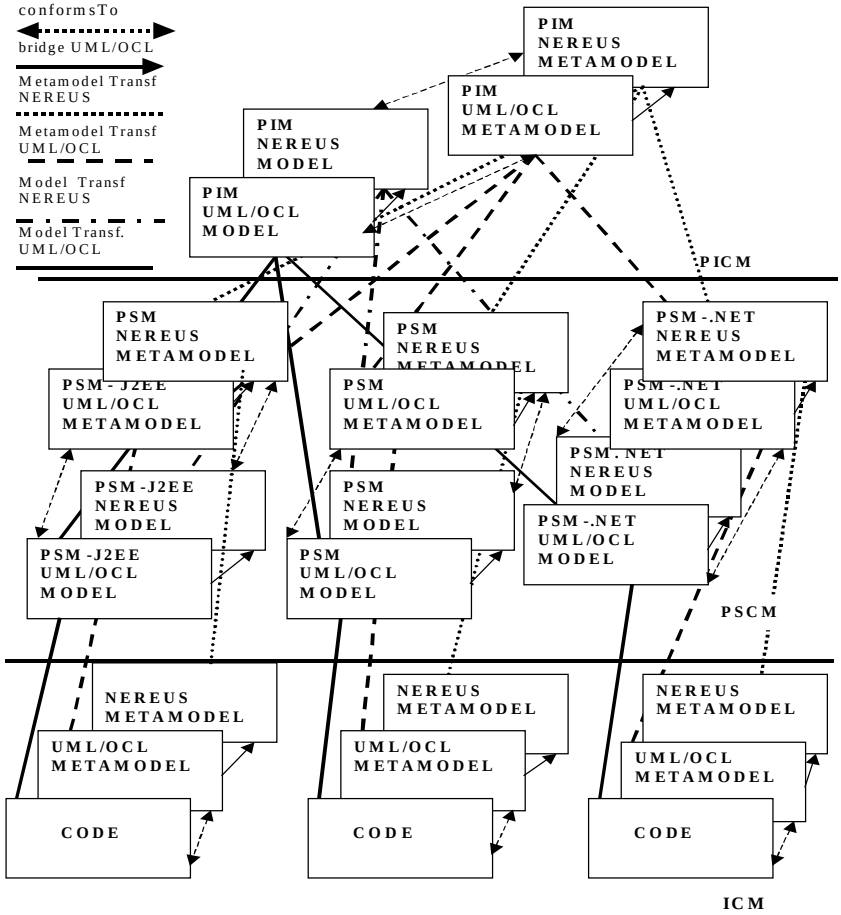


Fig. 1. A megamodel for MDA-based components

3 NEREUS, a Metamodeling Notation

To enable automatic transformation of a model, we need metamodels to be written in a well-defined language. The formalization of metamodels can help us to address MDA. We propose to specify UML static models, metamodels and meta-metamodels in the intermediate notation NEREUS that is independent of any specific formal language and can be translated into specific ones.

NEREUS consists of several constructions to express classes, associations and packages. It distinguishes clientship, inheritance and subtyping relations. The IMPORTS, INHERITS and IS-SUBTYPE-OF clauses express clientship, inheritance

and subtyping relations respectively. Subtyping is like inheritance of behavior, while inheritance relies on the module viewpoint of classes. A notion closely related with subtyping is polymorphism, which satisfies the property that each object of a subclass is at the same time an object of its superclasses.

NEREUS distinguishes deferred and effective parts. The DEFERRED clause declares new types or operations that are incompletely defined. The EFFECTIVE clause either declares new types or operations that are completely defined, or completes the definition of some inherited type or operation.

Operations are declared in the FUNCTIONS clause that introduces the operation signatures, the list of their arguments and result types. They can be declared as total or partial. NEREUS allows us to specify operation signatures in an incomplete way. NEREUS supports higher-order operations (a function f is higher-order if functional sorts appear in a parameter sort or the result sort of f). In the context of OCL *Collection* formalization, second-order operations are required. In NEREUS it is possible to specify any of the three levels of visibility for operations: public, protected and private. NEREUS provides the construction LET… IN. to limit the scope of the declarations of auxiliary symbols by using local definitions.

Several useful predefined types are offered in NEREUS, for example *Collection*, *Set*, *Sequence*, *Bag*, *Boolean*, *String*, *Nat* and enumerated types.

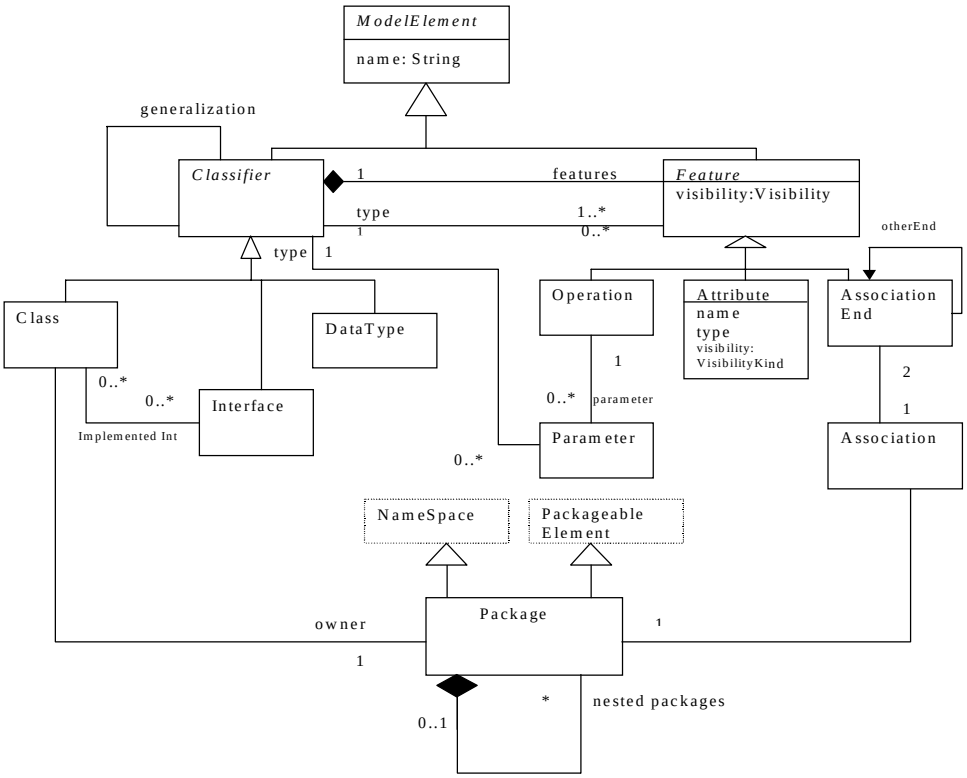
NEREUS provides a taxonomy of constructor types that classifies binary associations according to kind (aggregation, composition, association, association class, qualified association), degree (unary, binary), navigability (unidirectional, bidirectional) and, connectivity (one-to-one, one-to-many, many-to-many). The package is the mechanism provided by NEREUS for grouping classes and associations and controlling its visibility.

Fig. 2 depicts a simplified version of the UML metamodel and its NEREUS specification. The metamodel includes the core modeling concepts of the UML class diagrams, including classes, associations and packages. A detailed description of NEREUS may be found at [12].

4 A Bridge between UML and NEREUS

We define a bridge between UML class diagrams together with OCL invariants and pre- and postconditions into a NEREUS specification. The text of the NEREUS specification is completed gradually. First, the signature and some axioms are obtained by instantiating reusable schemes. Next, associations are transformed by using the constructor type Association. Finally, OCL specifications are transformed using a set of transformation rules. Then, a specification that reflects all the information of UML diagram is constructed.

Analyzing OCL specifications we can derive axioms that will be included in the NEREUS specifications. Preconditions written in OCL are used to generate preconditions in NEREUS. Postconditions and invariants allow us to generate axioms in NEREUS. The transformation process of OCL specifications to NEREUS is supported by a system of transformation rules.



PACKAGE Core **CLASS** ModelElement **DEFERRED** **TYPES** ModelElement **FUNCTIONS** name: ModelElement -> String … **END-CLASS** **CLASS** Classifier **IS-SUBTYPE-OF** ModelElement **DEFERRED** **ASSOCIATES** <<generalization>>, <<specialization>> <<has-Features>>, <<Classifier-Feature>>, <<Classifier-Parameter>> **TYPES** ModelElement **FUNCTIONS** … **END-CLASS**	**CLASS** Operation … **CLASS** Attribute … **ASSOCIATION** has-Features **IS** Composition-2 [Classifier : class1; Feature : class2; classifier : role1; features: role2; 1: mult1; 1..*: mult2; +: visibility1; +: visibility2] **END** **ASSOCIATION** Operation-Parameter **IS** Bidirectional-2 [Operation: class1; Parameter: class2; operation: role1; parameter: role2; 1: mult1; 0..*: mult2; +: visibility1; +: visibility2] **END** … **END-PACKAGE**

Fig. 2. A Simplified Metamodel and its NEREUS Specification

Fig. 3. shows a simple class diagram Person-Meeting. [10] analyzes this example and [11] describes in detail these translations. In this work we only remarks the translation of OCL into NEREUS.

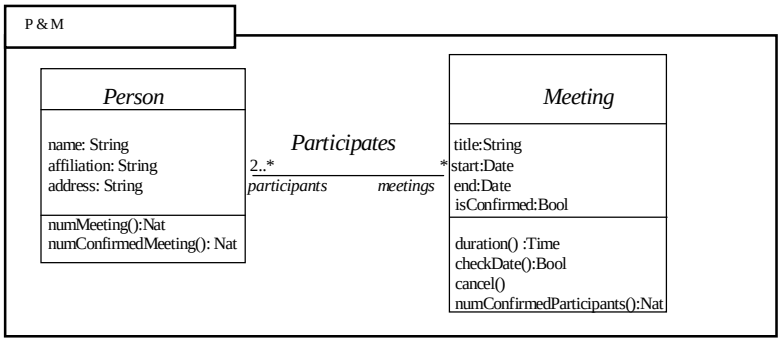


Fig. 3. An example: The Person&Meeting Package

An operation can be specified in OCL by means of pre- and post-conditions. *Self* can be used in the expression to refer to the object on which the operation was called, and the name r*esult* is the name of the returned object, if there is any. The names of the parameter *(parameter1,...)* can also be used in the expression. To refer to the value of a property at the start of the operation, one has to postfix the property name with "@", followed by the keyword "pre". For example, the OCL specification for an operation called *AddPerson* is translated as follows:

OCL	***NEREUS***
AddPerson (p:Person) **pre**:notmeetings->meetings@pre->including (p)	AddPerson: Participates (a) x Person (p) -> Participates **pre** : not includes (getMeetings (a), p) **Axioms** a:Participates; p:Person;… getMeetings(AddPerson (a,p)) = including (getMeetings (a), p)

Fig. 4 shows how to map OCL specifications of *Person&Meeting* onto NEREUS.

5 Integrating NEREUS with Algebraic Languages

In this section we examine the relation between NEREUS and algebraic languages using CASL (*Common Algebraic Specification Language*) as a common algebraic language [5].

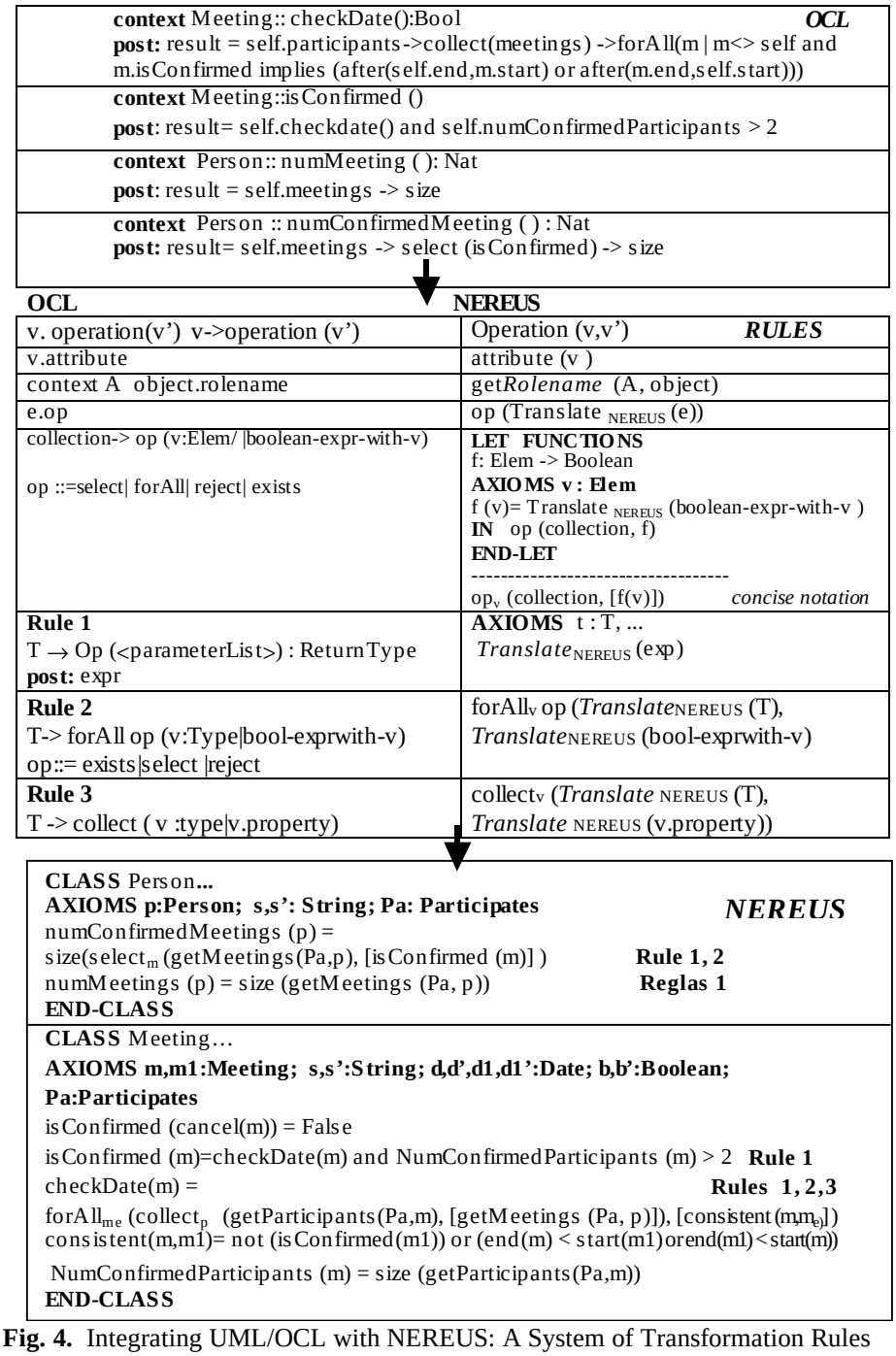


Fig. 4. Integrating UML/OCL with NEREUS: A System of Transformation Rules

CASL is an expressive and simple language based on a critical selection of known constructs such as subsorts, partial functions, predicates, first-order logic, and structured and architectural specifications.

Architectural specifications impose structure on implementations, whereas structured specifications only structure the text of specifications. CASL is supported by tools and facilitates interoperability of prototyping and verification tools.

We have defined the translation of NEREUS to CASL. A detailed description may be found at [12]. An interesting problem is how to translate associations. NEREUS and UML follow similar structuring mechanisms of data abstraction and data encapsulation. The algebraic languages do not follow these structuring mechanisms in an UML style. In UML an association can be viewed as a local part of an object. This interpretation can not be mapped to classical algebraic specifications which do not admit cyclic import relations.

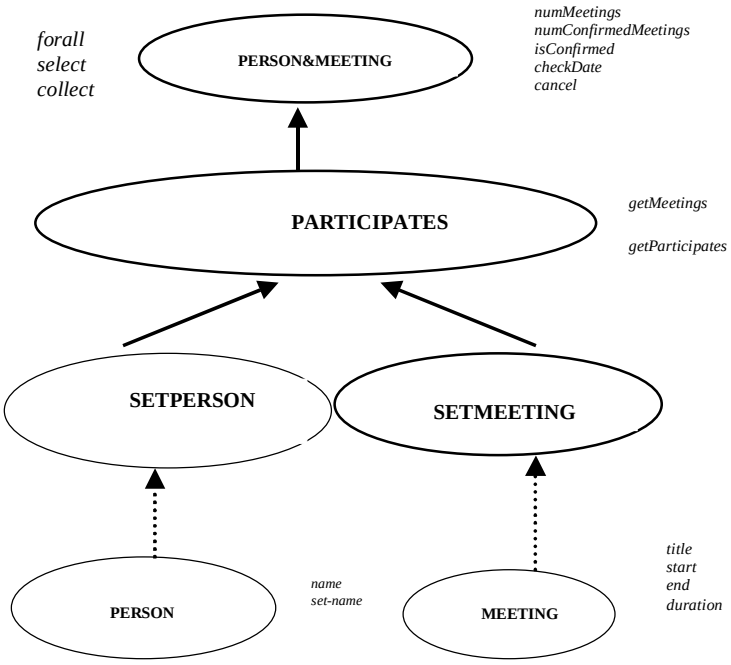
We propose an algebraic specification that consider associations belonging to the environment in which an actual instance of the class is embedded. Let *Assoc* be a bi-directional association between two classes called *A* and *B*, the following steps can be distinguished in the translation process:

Step1*:* Regroup the operations of classes *A* and *B* distinguishing operations local to *A*, local to *B* and, local to *A* and *B* and *Assoc*.
Step 2: Construct the specifications *A'* and *B'* from *A* and *B* where *A'* and *B'* include local operations to *A* and *B* respectively.
Step 3*:* Construct specifications *Collection[A']* and *Collection[B']* by instantiating reusable schemes.
Step 4: Construct a specification *Assoc* (with C*ollection [A']* and *Collection[B']*) by instantiating reusable schemes in the component *Association.*
Step 5*:* Construct the specification *AssocA+B* by extending *Assoc* with *A'*, *B'* and the operations local to *A'*, *B'* and *Assoc*

We exemplify these steps with the transformation of *Person&Meeting.* Fig. 5. depicts the relations among the specifications built in the different steps and, the final specification in CASL.

6 Related Work

Several metamodeling approaches have been proposed to Model-Driven development [1], [3], [6], [9], [16]. Currently a lot of research is performed in the direction of how to express model transformations [4], [7]. [2] propose an approach which uses metamodeling patterns that capture the essence of mathematical relations. The proposed technique is to adopt a pattern which models a transformation relationship as a relation or collections of relations, and encode this as an object model. In [14], it was defined an extension of a metamodeling language to specify mappings between metamodels based on concepts presented in [2].



spec PERSON&MEETING = PARTICIPATES
then ops
numMeeting : Participates x Person -> Nat
numConfirmedMeeting : Participates x Person -> Nat
isConfirmed : Participates x Meeting -> Boolean
numConfirmedParticipants: Participates x Meeting -> Nat
checkDate: Participates x Meeting -> Participates
select : Participates x Set[Meeting] -> Set[Meeting]
collect: Participates x Set[Person] -> Bag[Meeting]
pred forall: Participates x Set[Meeting] x Meeting
isConsistent: Participates x Meeting x Meeting
∀s : Set[Meeting]; m:Meeting; pa:Participates; p:Person; m:Meeting; sp:Set[Person]; bm: Bag[Meeting]
forall(pa, including(s,m),m1) = isConsistent(pa, m,m1) and forall(pa, s, m1)
select(pa, create-Meeting) = create-Meeting
select (pa, including (s, m)) = including(select(pa,s), m) **when** isConfirmed(pa,m)
else select (pa,s)
collect (pa, create-Person,s) = asBag (create-Person)
collect (pa, including (sp, p)) = asBag (including (collect (pa,sp), p))
numMeeting(pa, p) = size (getMeetings(pa, p))
isConfirmed (pa, m) = checkDate (pa,m) and NumConfirmedParticipants(pa,m)>2
numConfirmedMeeting (pa, p) = size (select (getMeetings (pa,p))
checkDate (pa, m) = forall (pa, collect (pa, getParticipants(pa,m), m)
isConsistent (pa, m, m1) = not (isConfirmed (pa,m1)) or (end(m) < start (m1)
or end (m1) < start(m))
numParticipantsConfirmed (pa, m) = size(getParticipants (pa, m))
end

Fig. 5. Integrating NEREUS with CASL

[15] compares and contrasts two approaches to model transformations: one is graph transformation and the other is a relational approach. [8] proposes a taxonomy for the classification of several existing and proposed model-to-model transformations approaches.

Currently, few tools provide limited support for MDA paradigm. Some of them are AndroMDA, OptimalJ and ArcStyler, among others [19]. The existing MDA-based tools do not provide sophisticated transformation from PIM to PSMs.

The following differences between our approach and some existing ones are worth mentioning. There are UML formalizations based on different languages that do not use an intermediate language. However, this extra step provides some advantages. NEREUS is a notation that can be used to mediate between UML and different formal languages. Any number of formal languages can be connected without having to write transformation systems from OCL to different formal languages. A formal specification clarifies the intended meaning of metamodels and helps to validate PIM models. NEREUS can provide a neutral basis for transforming PIM to PSMs. Also, intermediate specifications may be needed for refactoring and for forward and reverse engineering purposes based on formal specifications.

Although OCL is a textual language, OCL expressions rely on UML class diagrams, i.e., the syntax context is determined graphically. OCL does also not have the solid background of a classical formal method. Then, it is interesting to integrate UML/OCL with formal languages. Our approach avoids defining transformation systems as well as the formal languages being used. We define an only bridge between UML/OCL and NEREUS by means of a transformational system consisting of a small set of transformation rules that can be automated.

7 Conclusions

In this paper, we describe a uniform framework for Model Driven Development that integrates UML/OCL specifications with formal languages.

NEREUS would allow us to take advantage of all the existing theoretical background on formal methods using different tools in different stages of MDD. NEREUS could be used to validate initial models, whose quality will determine the quality of the transformed models. Rather than requiring that designers manipulate metamodels and formal specifications, we want to define foundations for MDA tools that permit designers to directly manipulate UML/OCL models they have created. However, meta-designers need to understand metamodels and metamodel transformations.

We are validating the megamodel through forward engineering, reverse engineering, model refactoring and pattern applications. In [11] we describe how to forward engineering UML static models to object-oriented code in a model driven fashion. We have defined MDA components of standard design patterns. We foresee to integrate our results in the existing UML CASE tools experimenting with different platforms such as .NET and J2EE.

References

1. Abmann, U. (ed.): Proc. of Model-Driven Architecture: Foundations and Applications, Linkoping University, Switzerland (2004)
2. Akehurst, D., Kent, S. A: Relational Approach to Defining Transformations in a Metamodel. In: Jezequel, J. M., Hussmann, H., Cook, S. (eds): Lecture Notes in Computer Science, Vol 2460, Springer-Verlag, Berlin Heidelberg New York (2002) 243-258
3. Atkinson, C., Kuhne, T.: The Role of Metamodeling in MDA. In: Bezivin, J., France, R. (eds): Proc. UML'2002 Workshop in Software Model Engineering (WiSME 2002). Technical Report, University of Nantes, www.metamodel.com/wisme-2002 (2002)
4. Bezivin, J., Jouault, F., Valduriez, P.: On the need for Megamodels. In: Bettin J. et. al (eds): Proc. Best Practices for Model-Driven Software Development (MDSD 2004), OOSPLA 2004, Vancouver, Canada (2004)
5. Bidoit, M., Mosses, P.: CASL User Manual- Introduction to Using the Common Algebraic Specification Language. Lecture Notes in Computer Science, Vol. 2900, Springer-Verlag, Berlin Heidelberg New York (2004)
6. Büttner, F., Gogolla, M. Realizing UML Metamodel Transformations with AGG. In: Heckel, R. (ed): Electronic Notes in Theoretical Computer Science, Vol 109 (2004) 31-42
7. Caplat, G., Sourrouille, J.: Model Mapping in MDA. In: Bezivin, J. France, R. (eds): Proc. UML'2002 Workshop in Software Model Engineering (WiSME 2002), http://www.metamodel.com/wisme-2002 (2002)
8. Czarnecki, K., Helsen, S.: Classification of Model Transformation Approaches. In: Bettin, J et al. (eds) Proc. OOSPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, www.oopsla.acm.org/oopsla2003 (2003**)**
9. Evans, a. Sammut, P. Willans, S. (eds): Metamodeling for MDA Workshop, York, England (2003)
10. Favre, L.: A Formal Mapping between UML Static Models and Algebraic Specifications. In: Evans, A. France, R., Moreira, A., Rumpe, B. (eds): Practical UML-Based Rigorous Development Methods-Countering or Integrating the eXtremist, Lecture Notes in Informatics (P 7) SEW, GI Edition, Alemania (2001) 113-127
11. Favre, L.: Foundations for MDA-based Forward Engineering. Journal of Object Technology (JOT), Vol 4, no 1 (2005) 129-154
12. Favre, L: The NEREUS Language. Technical Report, INTIA; Universidad Nacional del Centro, Argentina (2003)
13. Gogolla, M. et. al. (ed.): Proceedings WISME 2004, Workshop in Software Model Engineering, Lisboa, Portugal (2004)
14. Haussmann, J.: Relations-Relating metamodels. In: Evans, A., Sammut, P. , Williams, J. (eds): Proc. Metamodeling for MDA. First International Workshop, York, UK (2003) 147-161
15. Kuster, J., Sendall S., Wahler M.: Comparing Two Model Transformation Approaches. In: Bezivin, J. et. al (eds): Proc. OCL&MDE'2004, OCL and Model Driven Engineering Workshop , Portugal (2004)
16. MDA. The Model Driven Architecture, OMG. www.omg.org/mda (2004)
17. OCL Specification. Versión 2.0. Documento ptc/03-03-14. www.omg.org (2004)
18. OMG. Object Management Group Documents. www.omg.org (2004)
19. UML Tools. In www.objectsbydesign.com/tools/ (2004)