Semantic Equations for Formal Models in the Model-Driven Architecture

Paulo Barbosa[1], Franklin Ramalho[1], Jorge Figueiredo[1], Anikó Costa[2], Luís Gomes[2], and Antônio Junior[1]

[1] Universidade Federal de Campina Grande, Brasil
{paulo,franklin,abrantes,antonio}@dsc.ufcg.edu.br
[2] Universidade Nova de Lisboa, Portugal
{akc,lugo}@uninova.pt

Abstract. Semantic equations are important pieces of the denotational semantics approach that enable the translation of syntactic constructs of models to semantic domains. In a previous work, we have extended the Model-Driven Architecture (MDA) four-layer proposal of architecture in order to incorporate formal semantics to its artifacts. Thus, semantic equations are key elements in this extended architecture. In this paper, we propose an approach to incorporate semantic equations described by model transformations in the MDA infrastructure. As a result, we automatically generate semantic models from the syntactic models involved in the MDA transformations. Therefore, the semantic equations play an important role in the MDA architecture since they allow the construction of tools for automated reasoning for several purposes, such as: (i) verification of semantics preservation in model transformations; (ii) choice of the best format to represent meaning and behavior of models and (iii) automatic generation and production of quality compilers. For instance, we claim that in the scenario of model transformations involving concurrent models described using Petri nets, semantic concepts can be more easily and appropriately handled in the MDA architecture. We have evaluated our approach by applying semantic equations to Petri nets models in a project for embedded systems codesign.

Keywords: Model-Driven Architecture, Petri nets, Denotational Semantics, Transformations, Semantic Equations.

1 Introduction

In the Model-Driven Architecture (MDA) [1] [2], a model is an abstract or concrete representation of a domain. They are described by metamodels that express the elements of the language. Model transformations process these elements through a set of definition rules that describe how to generate an output model from an input model. Although MDA promises to overcome important gaps in the software engineering, it has not specified ways to ensure that its artifacts are correctly represented. Particularly, the lack of a formal representation of the involved models in the four-layer MDA architecture leads to undesirable situations of ambiguity and low reliability.

In [3], we have proposed an extension of the four-layer MDA architecture in order to incorporate formal semantics in its infrastructure. We introduced semantic metamodels and models, and other artifacts to verify semantics equivalence between the input and output models involved in MDA transformations and to prove properties about them. This proposal was evaluated in the concurrent systems domains, through models represented using Petri nets [4] in a project of embedded systems co-design [5].

Assuming the importance of formal semantics in the modeling domain, to apply it to a model it is necessary the assignment of meanings to its sentences or components.

This needs to be treated in the proposal of the extension of the MDA architecture. This assignment should be given by a mathematical model that represents every possible computation in the language that describes the model. It is expected with the semantic description of a model the following benefits: (i) the development of partial evaluation techniques for interpreting and compiling models in the MDA architecture; (ii) the automatic generation of compiler semantics definitions; (iii) a foundation for understanding and evaluating the design issues; and (iv) a valuable reference for transformations that involve this model. In this sense, the formal definition is necessary to guarantee the preservation of properties in the output model after a transformation, which is the main goal of a PhD thesis that guides this work.

The denotational semantics [6] is one of the most employed approaches to provide formal semantics. It is specially addressed for the case in which the necessity of constructing mathematical objects to represent meaning of models emerges, as is in this work. In this approach, the meaning is given by functions that map syntactic elements to mathematically well-defined sets. This mapping is usually named as semantic equations.

Although semantic models are considered as important pieces for a formal architecture, in the extension of the MDA architecture they make sense only in the case of being automatically generated from the syntax. Most approaches based on semantic models depend on the expertise of a domain engineer, demanding costs, low reliability and very imprecise specifications as representations of what of most precise a model should have [7].

In this paper, we claim that the extraction of semantic models in MDA must be responsibility of the semantic equations, following declarative formal rules using automated tools according to the MDA standards. Semantic models can be described by a metamodel with the same expressive power as the metalanguage of denotational semantics. This fills the necessity of compiler generators guided by semantics as a unified framework. Denotational semantics specifications may have the format of a MDA transformation, providing this unification as mathematical objects and turning modeling languages closer to this metalanguage of formal semantics.

The structure of this paper is as follows. Section 2 discusses how this work can contribute to innovation in the specific MDA technology. Section 3 and 4 presents the main background issues necessary for understanding this work, namely our previous proposal of the extending the MDA architecture and the denotational semantics concepts involved. Section 5 gives details about our proposed approach to specify semantic equations illustrating and evaluating for specific models. Section 6 discusses related works. Finally, Section 7 discusses about the benefits and possible drawbacks of the presented work, and the final remarks.

2 Contribution to Technological Innovation

In this work, we propose and discuss an approach to introduce semantic equations as formal specifications in the MDA architecture. Considering MDA transformations as the pivotal element to specify semantic equations, some issues emerge to be answered, as pragmatic aspects of the transformation language and some theoretical aspects related to computability. These issues are addressed and discussed in this work.

We evaluate this proposal with a well-established and appropriate extraction from the syntactic constructs to semantic domains in the concurrent domain with Petri nets models. This allowed the verification of semantics preserving transformations as well as an on-the-fly occurrence graph generation system for it. Moreover, with the proposal of a representation to semantic equations, semantic properties of models can be checked for several purposes.

3 The Extended MDA Four-Layer Architecture

The current MDA architecture, shown in Figure 1, consists of four layers: (i) M0 involves executable representations of models as code; (ii) M1 describes the concrete syntax representation of a given model; (iii) M2 provides the metamodel which serves as a grammar to describe the model syntax developed at the layer M1; and (iii) M3 describes the layer M2 by using MOF, the Meta-Object Facility [2]. Since MOF describes itself, it is not required further metamodels. The model transformations are able to automatically generate output models from input models at the layer M1. They are defined in terms of metamodel descriptions and cope only with syntactic/structural aspects.

Our proposal of an extended MDA architecture for incorporating formal semantics is shown in Figure 2 as gray modules at the layers M1 and M2. Semantic equations, which are the focus of this paper is highlighted and will be discussed. Briefly, the other new inserted parts are:

- Semantic metamodel, which defines (i) the state configuration of a model or a program according to the semantic specifications; and (ii) the dynamic semantics capturing the state infrastructure of language constructs as well as how they use or change an existing state. It serves as a concretization for theories of formal semantics.
- Semantic model, which instantiates the state configuration metamodel. Since semantic metamodels represent theories of formal semantics, semantic models are representations according to these theories.
- Simplification rules, which are able to perform the inference and computation of state configurations. They instantiate the dynamic semantics. These rules are implemented according to the chosen computation mechanism.
- Formal checker, which is complementary to the simplification rules, also instantiates the dynamic semantics, and it is required to prove properties about the equivalence of the verified input and output models.

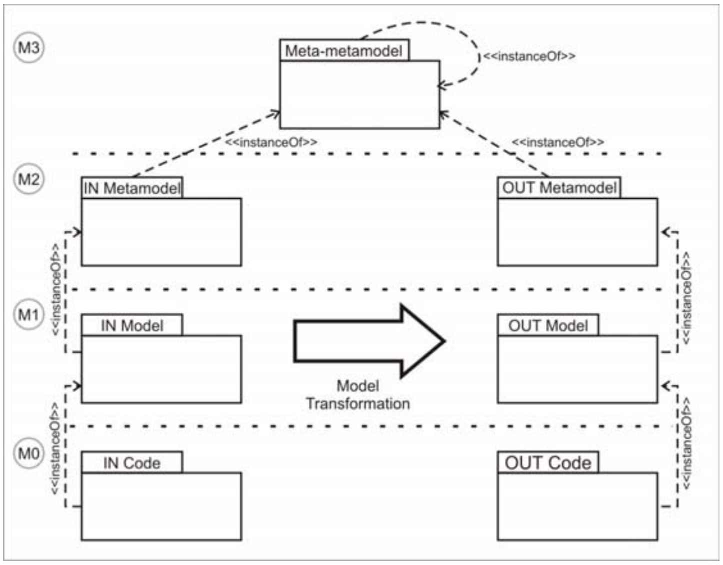

Fig. 1. The current MDA Architecture

The role of semantic equations in this architecture is to define mappings, from the language's abstract syntax structures to meanings drawn from semantic models. They are illustrated as the small arrows from the syntactic models to the semantic models in the layer M1.

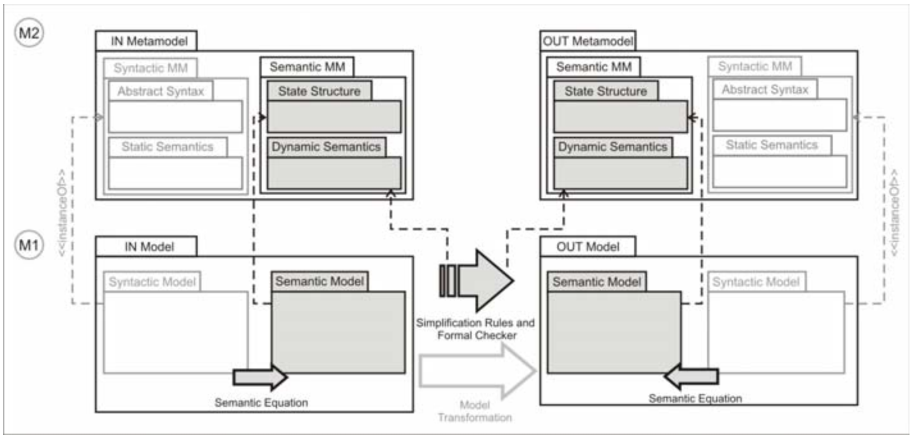

Fig. 2. The extended architecture to incorporate formal semantics

4 Denotational Semantics for Petri Nets

The main aim of denotational semantics is to provide a proper mathematical foundation for reasoning about programs and programming languages. It is based on a set of denotations, i.e., a function with arguments that represent expressions before and after executing. Denotations can inductively be defined by using any formal mapping to specify how the denotations of components are combined.

To give the denotational semantics of a language, a suitable collection of meanings is necessary. The most used theory is called Domain Theory, which employs structured sets, called domains, and their operations as data structures for semantics. These domains plus their operations constitute a semantic algebra. According to [8], the semantic algebra: (i) clearly states the structure of a domain and how its elements are used by the functions; (ii) modularizes and provides reuse of semantic definitions; and (iii) makes analysis easier.

We have chosen the Petri nets formalism as the description language of the involved models because this language is already well-accepted in the MDA community [9] and has a good support to formal semantics. It was designed as a graphical and mathematical formalism for modeling concurrent systems. Moreover, this work is inserted in the context of a project that proposes the use of Petri nets for the co-design of embedded systems.

We take the nets analyzed in [10] which are shown in Figure 3 and refers to the model of a parking lot controller when considering a centralized implementation or a distributed implementation relying in two components (synchronized through transition firing) after applying the Splitting Operation.

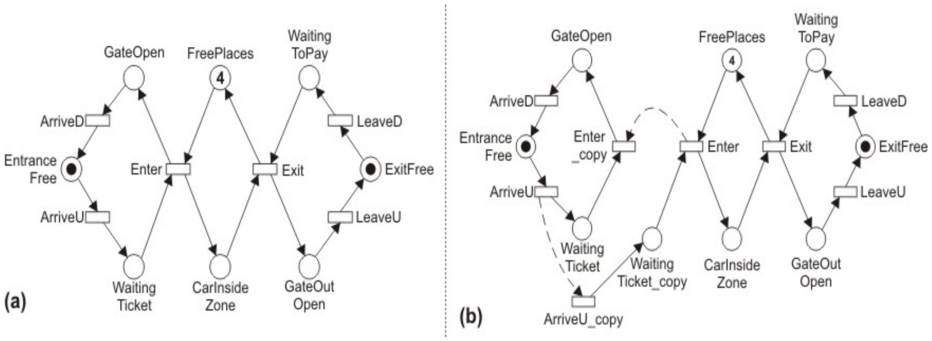

Fig. 3. Two nets that represent models before and after the Splitting Operation

In [11], Meséguer and Montanari use the category theory to claim that a Petri net model is to be translated to an ordinary, directed graph equipped with two algebraic operations. These operations correspond to the sequential and parallel composition of transitions (representing behavior), with a distributive law between these compositions.

The model has to provide the generators of the algebraic structure which the responsibility of the automatic mapping is given to the semantic equations.

The understanding of concurrency in terms of algebraic structures over graphs is very important for Petri nets modelers. It has the potential of unifying several views of semantics formalisms. Moreover, this structure can be manipulated by formal methods tools to be employed in several sorts of analysis and satisfying some requirements that are hard to solve by most editors and tools available for this domain. As an example, several formal tools could afford to the state space explosion problem, by providing an on-the-fly way of reasoning over occurrence graph structure of the Petri net model if it is seen as this algebraic structure.

By using more specific terms, this ordinary graph has a set of nodes as a commutative monoid $S^{\oplus}$ generated by the set S of places and the empty marking as identity. The sum **+** of transitions represents the parallel firing of transitions, and the sequential firing of transitions is denoted by the operation. By the guarantee of the closure property, we are representing computations through simple transitions.

For example, we have that the next code fragment is a representation of the semantic model for the Maude System [12] from the net before the Splitting Operation of Figure 3 as an illustration of the role that semantic models can play after extracted from the semantic equations. The Commutative operation + is present in line 7 with the specified properties. This semantic model may be used for several verification purposes, such as model-checking or theorem proving.

```
  1: mod INPUT_NET is
  2:  sorts Place Marking .
  3:  subsort Place < Marking .
  4:  ops GateOpen EntranceFree WaitingTicket CarInsideZone
  5:   GateOutOpen ExitFree WaitingToPay FreePlaces : -> Place.
  6:  op empty : -> Marking.
  7:  op _+_:Marking Marking->Marking [assoc comm id:empty].
  8:  rl [ArriveD]:GateOpen=>EntranceFree.
  9:  rl [ArriveU]:EntranceFree=>WaitingTicket.
  10: rl [Enter]:WaitngTicket + FreePlaces=>CarInsdeZone +
  GateOpen.
  11: rl [Exit]:CarInsdeZone + WaitingToPay=>GateOtOpen + Free-
  Place.
  12: rl [LeaveU]:GateOutOpen=>ExitFree.
  13: rl [LeaveD]:ExitFree=>WaitingToPay.
  14:endm
```

5 MDA Semantic Equations

In order to incorporate semantic equations into the extended MDA architecture, it is necessary to cope with pragmatic and formal issues. The former concerns requirements related to the language and tools that will represent semantic equations whereas the latter deals with theoretical aspects to ensure composition and recursion as is required by denotational semantics specifications.

Focusing on the pragmatic issues, there are three requirements that must be observed on the language used to represent semantic equations:

i. It must be able to represent the functional characteristics of classical definitions for denotational semantics. Supporting the declarative paradigm allows pattern matching that is essential to describe functional operations. Moreover, this seems also necessary because the language that will describe the codomain of semantic equations as in denotational semantics is completely based on functional constructs, such as functions definitions and applications.

ii. It must support the representation of actual features of most programming models, representing aspects of programming and modeling languages. Concepts such as sequentiality, concurrency, non-determinism and an infrastructure to represent states are present whatever be the domain of the semantic equations application.

iii. It must manipulate and execute properly the semantic equations. This will enable the integration with other formal methods tools. A powerful toolset of the language to support this seems to be essential to our solution.

On the other hand, there are two theoretical aspects of the language that must be addressed:

iv. It must be fully (or partially) compositional. The understanding of composite expressions as transformations rules makes the proof of properties about semantics easier, allowing the translation of syntactic pieces to a semantic metalanguage represented as a semantic algebra metamodel. This represents a challenge but also a useful improvement because there are usually propositions, axioms and theorems that have been developed for this metalanguage. Finding such a compositional language allows us to reason using induction over the grammar of the modeling language, since this grammar can be described by metamodels.

v. It must allow recursion over the grammatical structure (i.e. the metamodel) of the language it interprets. In other terms, it must enable to extract the current state of the model in terms of a meaningful representation (i.e. the semantic model). This is made by carrying a structure representing the environment of a program or of a model. When the input model (i.e. syntactic model) is given to the engine, the transformation explores it at the top-level term, creating any necessary context into an environmental structure, passes this structure into a recursive call for each of the subterms, and provides the terms able to be analyzed based on what it gets back from the mapping that represents the semantic equations.

This work characterizes semantic equations as, essentially, MDA transformations. In order to prove this, we choose a transformation language and MDA standards to accomplish the presented pragmatic and formal issues. This was evaluated through a case study that we conducted for the mapping from syntactic structures to semantic domains. For this, we have chosen the ATL language [13], because this language is able to satisfy the mentioned requirements. Moreover we discuss this justification by using Petri nets models.

Concerning the previously pragmatic addressed issues, we have a discussion about each topic. About (i), classical definitions of denotational semantics are present requiring a declarative approach for the definition of functional algebraic concepts. In (ii), we have that imperative fragments are necessary in when defining semantic equation, dealing with common concepts such as variables and sequencing. Finishing the first part of issues, we have that in (iii), ATL allows the manipulation of formal semantics concepts using an editor and an engine tool able to execute these specifications according to [13]. Concerning the theoretical issues, we see that in (iv), the compositionality of ATL enables the definition of the mapping into simpler parts, treating each algebraic structure as mathematics does and combining these rules for the whole. Finally the issue (v) is answered because ATL searches recursively in the language metamodel, dealing with the compositionality of the subterms, converting abstract syntax structures into the predefined algebraic structures responsible for the representation of meaning as semantic models.

6 Related Works

Regarding to related works we concentrate on the expected benefits of having semantic models using MDA standards through semantic equations. [14] proposes the use of Petri nets to simulate the behavior of models in MDA. The models are described in UML state chart Diagrams and then converted to a Colored Petri nets representation. In this case, Petri nets models act as semantic models. The work presented in [15] proposes the MOF formalization reusing the category theory modularization concepts from algebraic specification languages. This provides a graph-transformation-based formalization of MOF, upgrading MOF with new interfaces and a composition operator based on graph morphisms. It is the same idea reused here, however is a proposal for the meta-metalanguage of the MDA framework, and not for a specific domain as we made for Petri nets. In [16] the authors propose a novel approach for the definition of a formal semantics to languages based on the Alloy language. It intends to turn semantics definitions easier, encouraging the adoption of formal techniques. This work advocates that its approach provides two main advantages: uniform notation and a mechanism for analysis. Finally, [17] concentrates in some specific problems commonly found when using MDA such as refactorings and extraction of meanings of UML models. This work makes use of mathematical techniques, specially the lattices theory, to discover abstractions from a set of formal objects. This seems very useful for providing sound refactorings. The similarity with our work is that semantic equations can also be viewed as ATL transformations, when mapping a UML diagram to a formal context (the semantic model).

7 Final Remarks and Future Works

It is already known that MDA's artifacts and processes lack a semantic model, and that there is no way to ensure that its transformations preserve the semantics. The use of an underlying semantic model for verification purposes is a fundamental step in the MDA transformations, in order to have a formal definition of the involved artifacts in the transformation and be able to check its correctness by comparing whether the input and output models are consistent according to the semantic model definition.

Semantic models are also suitable for a compiler generation system for MDA. Currently, the models and metamodels built in this infrastructure can deal only with abstract and concrete syntax terms. Semantic equations are able to map to the denotational definition from a programming or modeling language. Concepts like semantic domains and semantic equations are specifications and the concrete implementation of a MDA compiler is encouraged. It enables dealing with semantic concepts such as scope, declarations, functions, type inference and type checking, control flow, exceptions and much more. Thus, the suitability of a semantic model is a promising step towards the automatic generation and production of quality compilers and processors.

We intend to analyze our approach in other scenarios than Petri nets modeling. Although each scenario have particular properties to be considered when providing semantic models by means of semantic equations, we advocate that our approach deserves special attention in all cases since: (i) the provided semantic metamodel may be reused; (ii) semantic models are required in any scenarios (iii) the pragmatic and theoretical issues raised in this work must be addressed.

Acknowledgments. This work was partially supported by the collaborative project Verificação Semântica em Transformações MDA Envolvendo Modelos de Redes de Petri (MDA-VERITAS - Semantic Verification in MDA Transformation using Petri nets models), funded by FCT - Fundação para a Ciência e a Tecnologia (Portugal) and CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (Brazil) (http://www.uninova.pt/~mda-veritas).

References

1. Miller, J., Mukerji, J.: MDA Guide Version 1.0.1. Object Management Group, OMG (2003)
2. OMG: Object Management Group (2008), `http://www.omg.org`
3. Barbosa, P., Ramalho, F., Figueiredo, J., Junior, A.: An Extended MDA Architecture for Ensuring Semantics-Preserving Transformations. In: Proceedings of 32nd Annual IEEE Software Engineering Workshop (2008)
4. Girault, C.V.R.: Petri Nets for Systems Engineering. In: XV, 607 p., Hardcover (2003)
5. Gomes, L., Barros, J.P., Costa, A., Pais, R., Moutinho, F.: Formal Methods for Embedded Systems Co-design: the FORDESIGN Project. In: ReCoSoC 2005- Reconfigurable Communication-centric Systems-on-Chip - Workshop Proceedings (2005)
6. Scott, D., Strachey, C.: Towards a Mathematical Semantics for Computer Languages. In: Proceedings of the Symposium on Computers and Automata. Microwave Research Institute Symposia Series, vol. 21 (1971)
7. Hausmann, J.: Dynamic Meta Modeling: A Semantics Description Technique for Visual Modeling Techniques. PhD thesis, University of Paderborn (2005)
8. Schmidt, D.A.: Denotational Semantics: a Methodology for Language Development. William C. Brown Publishers, Dubuque (1986)
9. Hillah, L., Kordon, F., Petrucci, L., Tréves, N.: Model Engineering on Petri nets for ISO/IEC 15909-2: API Framework for Petri Net types metamodels. Petri Net Newsletter, 22–40 (2005)

10. Barbosa, P., Costa, A., Ramalho, F., Figueiredo, J., Gomes, L., Junior, A.: Checking Semantics Equivalence of MDA Transformations in Concurrent Systems. Journal of Universal Computer Science, J.UCS (to appear, 2009), `http://www.jucs.org/jucs`
11. Meseguer, J., Montanari, U.: Petri nets are Monoids: a new Algebraic Foundation for Net Theory. In: Proceedings of the Third Annual IEEE Symposium on Logic in Computer Science (LICS 1988), pp. 155–164. IEEE Computer Society Press, Los Alamitos (1988)
12. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.F.: Maude: Specification and programming in rewriting logic. Theoretical Computer Science (2001)
13. Bezivin, J., Breton, E., Valduriez, P., Dupr, G.: The ATL Transformation-Based Model Management Framework. Technical report, produced from IRIN (2003)
14. Hu, Z., Shatz, S.M.: Mapping UML Diagrams to a Petri Net Notation for System Simulation. In: SEKE, pp. 213–219 (2004)
15. Weisemöller, I., Schürr, A.: Formal Definition of MOF 2.0 Metamodel Components and Composition. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 386–400. Springer, Heidelberg (2008)
16. Kelsen, P., Ma, Q.: A lightweight approach for defining the formal semantics of a modeling language. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 690–704. Springer, Heidelberg (2008)
17. Arévalo, G., Falleri, J.R., Huchard, M., Nebut, C.: Building abstractions in class models: Formal concept analysis in a model-driven approach. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 513–527. Springer, Heidelberg (2006)