Mapping Models Between Different Modeling Languages [⋆]

E. Domínguez[1], A. L. Rubio[2], and M. A. Zapata[1]

[1] Dpto. de Informática e Ingeniería de Sistemas.
Facultad de Ciencias. Edificio de Matemáticas.
Universidad de Zaragoza. 50009 Zaragoza. Spain.
{noesis, mazapata}@posta.unizar.es

[2] Dpto. de Matemáticas y Computación. Edificio Vives.
Universidad de La Rioja. 26004 Logroño. Spain
arubio@dmc.unirioja.es

Abstract. In this paper we show a process for defining a translation between two different modeling languages that allows to map a model expressed in a modeling language into a model expressed in the other modeling language. This mapping construction process is tackled from a metamodeling perspective and it is applied to fragments of OML and UML modeling languages.

1 Introduction

The Unified Modeling Language (UML) [18] is more and more widely accepted as a standard notation for object oriented software modeling. Since other object oriented modeling languages are currently being used in different application domains, there is a great interest in analyzing the similarities and differences between UML and other non–UML languages [1, 10]. Sometimes these comparisons are carried out in an informal way and in other cases a precise mapping from the models of a modeling language into models of the other modeling language is given [13, 17]. The problem is that these mappings are particular solutions defined between two specific languages, so they cannot be reused for defining translations between other modeling languages.

An approach that can be used for obtaining a general framework for model translation is that of metamodelling, since it has been proven that the use of metamodels improves rigor and facilitates system integration and interoperability [14]. The issue of defining model mappings using a metamodelling approach has been faced in the literature in different contexts, and with different metamodelling perspectives. For instance, within systems interoperability context, Nicolle et al. [16] propose the definition of only one metamodel, which, like an ontology, must generically represent the building blocks of data models. Grundy and Venable [9] deal with integration of different notations, achieving the interoperability by means of a mapping which is based on the construction of a metamodel for each one of the involved languages and a metamodel integrating the initial ones. As another example, within the COMMA project [11], an object-oriented core metamodel is proposed, so that the COMMA-compliant methodologies can be derived from it, allowing a smooth translation of the core concepts to be done.

A metamodelling perspective is also considered in [3], where we have proposed a concrete method for conducting the construction of a mapping between models of different modeling languages, using the Noesis technique as metamodeling language [5]. This approach also proposes to construct a metamodel for each of the involved languages, but unlike the approaches cited, our proposal consists of defining a chain of intermediate metamodels in order to bring together the initial ones. One of the main advantages of this perspective is that the complex problem of assuring that semantics are preserved in the mapping is delimited to simpler steps, in which other existing transformations (for instance refactoring transformations [19]) can be used.

[⋆] This work has been partially supported by DGES, projects TIC2000-1368-C03-01 and PB98-1621-C02-01

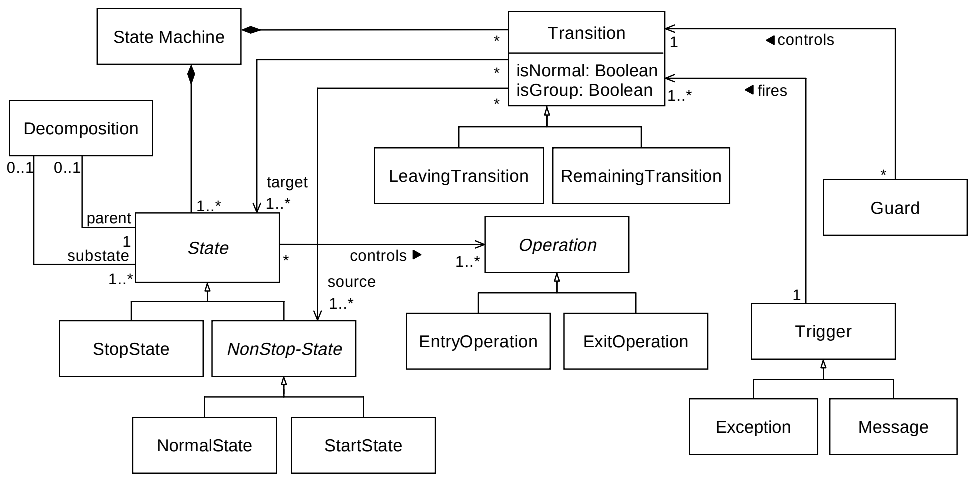

Fig. 1. UML metamodel of OML State Machines

The idea of this paper is to show how the mapping construction process proposed in [3] can be used for constructing, in a general way, well–defined translations between UML and non–UML languages. As a result, the method can be a helpful tool for analyzing the relationship of UML to other languages. Furthermore, in this paper we follow the UML definition style and for this reason we will use the UML itself as metamodeling language (instead of Noesis). This will prove, in a practical way, the independence between the proposed method and the chosen metamodeling language.

The paper is organized as follows: in the next section the mapping construction process is presented (illustrating it by means of an example) and, finally, conclusions are given.

2 Mapping Definition Process

In order to compare UML with other non–UML languages, we propose to bring into play the notion of mapping (or translation). Given two modeling languages L and L', a *mapping* refers to a method which allows a model of L' to be determined starting from a model of L[1]. The problem is that, in general, the definition of a mapping is a complex task. As a way of facilitating this process in [3] we proposed a mapping definition method within a metamodeling approach.

This method is based on the idea of defining a mapping starting from two metamodels (one for each of the languages), in the belief that this approach facilitates the mapping construction process. To be precise, within our proposal, the construction of a mapping between a language L, with metamodel M, and a language L', with metamodel M', consists of considering a finite chain of metamodels $M{=}M_1$, M_2, ...,$M_n{=}M'$ so that: 1) on the one hand, two consecutive metamodels M_i, M_{i+1} must differ slightly in order to make the development of a mapping between their models easier; 2) on the other hand, the composition of these easier mappings provides a complete mapping, which must fulfill our requirements. In order to construct this sequence of metamodels a five–step process is proposed. A noteworthy advantage of this multi–step process perspective is that it allows to delimit the difficulties in assuring that semantics are preserved in the translation. For instance, as it will be shown, in the third step the initital metamodels are approached by means of transformations that have been proved to be correct with respect to the semantics [2, 7, 15, 19].

As we have said before, in [3] we proposed to use the Noesis technique as metamodeling language, but in this paper we are going to use UML as the language for constructing the

[1] When the two languages are the same the mapping is called transformation.

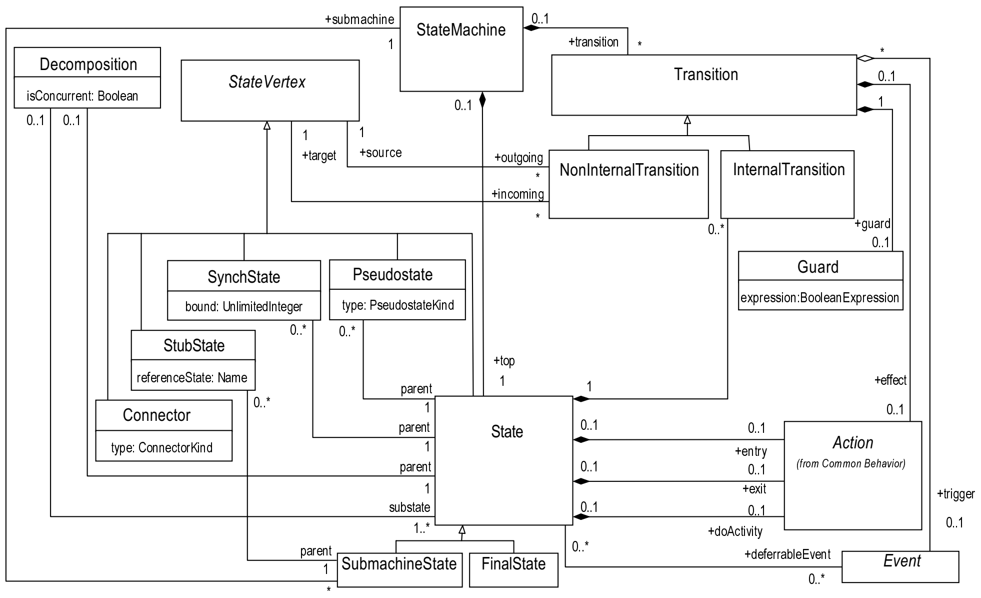

Fig. 2. UML metamodel of UML State Machines

metamodels (and so we also prove that the method can be used whichever it was the chosen metamodeling language). Therefore, the proposed mapping construction process must be applied after two UML metamodels are given.

In order to illustrate each step of the process we will present, as a practical application, the construction of a mapping from the state machines of OML [8, 12] into the state machines of UML [18]. Firstly a UML metamodel of each of these modeling languages has to be given. We have constructed a metamodel of the OML state machines (Figure 1) taken into account the OML metamodels proposed in [8] and [12] (but using UML as metamodeling language instead of OML). On the other hand, in [4], we propose a metamodel of UML state machines which appears in Figure 2. Once the two metamodels are defined, the steps for the construction of a mapping must be followed.

Step 1. Setting the mapping context. The context in which the mapping is meaningful is determined in this step, that is to say, it must be analyzed if a mapping can be defined for every model of the source metamodel. In order to do this, the ranges of application of the involved metamodels must be compared. In our particular example every OML state machine can be mapped into a UML state machine, but in general, if the ranges of application of the involved metamodels are different can be necessary to restrict the context.

Step 2. Understanding the similarities and differences. The goal of this step is to understand the similarities and differences between the two given modeling languages with the aim of getting a general perspective of the mapping that will be determined in the following steps. In order to do this, besides the published comparisons about the involved languages, the comparison of the two metamodels can be useful since they reveal aspects which can go unnoticed in informal comparisons [6]. For instance, the main concepts (state, transition, operation, trigger) of the OML State Machines metamodel also appear (some of them with different names) in the UML State Machines metamodel, but the subclasses are different so that this aspect has to be taken into account during the mapping construction process. Furthermore, it is very important the fact that the notion of concurrent decomposition is not considered within the OML state machines.

Step 3. Approaching the initial metamodels. In this step, starting from the two initial metamodels, increasingly more similar intermediate metamodels are defined. The idea

Table 1. Class diagrams transformations in the OML State Machines Metamodel

1. *GeneralizeAssociationEnd* (NonStop–state, source, State)
2. *RemoveClass* (NonStop–state)
3. *AbstractToConcrete* (State)
4. *RemoveClass* (NormalState)
5. *SplitAssociation* (Controls, {Entry, Exit})
6. *AbstractToConcrete* (Operation)
7. *RemoveClass* (EntryOperation)
8. *RemoveClass* (ExitOperation)
9. *RemoveAttribute* (Transition, isNormal)
10. *RemoveAttribute* (Transition, isGroup)
11. *RemoveClass* (Exception)
12. *RemoveClass* (Message)

is to smooth away their differences applying UML class diagrams transformations that are known to be correct with respect to the semantics. There are several papers (see [7, 15, 19]) that propose UML class diagrams transformations taking into account their semantics. These transformations, in general, allow a class, attribute, method or association to be added, moved or removed when the diagram holds several predefined constraints. Furthermore, other more complex transformations such as generalization, merge, in line and split [2] have also been proposed. Following the ideas proposed in the different above–mentioned papers we have applied a sequence of transformations to the OML State Machines metamodel (Table 1) obtaining as a result the equivalent metamodel that appears in Figure 3.

Step 4. Determining specific mappings. When it is considered that the metamodels cannot be approached any further by means of UML diagrams transformations that have been proposed in the literature, then ad hoc mappings have to be defined (when two or more mappings are defined the additional intermediate metamodels have to be determined). In this step the particular semantics of the involved techniques has to be taken into account in order to construct the specific mapping. In our example we can establish pairs of synonym concepts – (state, state), (operation, action), (trigger, event), etc – between the metamodels achieved in the previous step so that the instances of one concept can be considered as instances of the synonymous concept. This simple mapping can be defined for all the concepts except for the transition concept. In this case, each transition of the OML state machine (which has several source and target state vertices) has to be mapped into several transitions (with only one source and one target state vertex) adding the necessary fork and junction connectors. Finally, the different concurrent components of the OML state machine must be determined in order to establish the corresponding concurrent decompositions in the UML state machine. We are aware of this last step of the mapping from an OML state machine into a UML state machine is not obvious and that it must be analyzed and determined in a more precise way. This study remains an ongoing work.

Step 5. Determining the complete mapping. Finally, the global mapping from the source metamodel to the target one is defined as a suitable composition of the mappings obtained in the two previous steps.

3 Conclusions

We have shown practically how the mapping construction process we proposed in [3] can be applied in order to define a translation between UML and non–UML modeling languages. In particular, this process can help to delimit the difficulties in assuring that semantics are preserved in the translation. At the same time, the independence between the method and

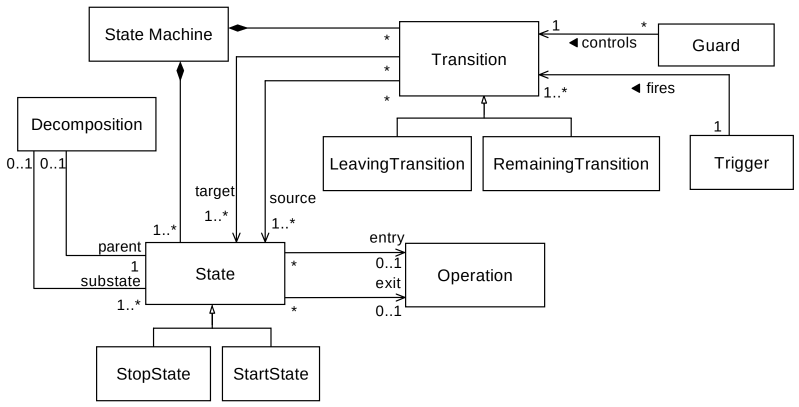

Fig. 3. Modified UML metamodel of OML State Machines

the metamodeling language has also been proven. As a future work the utility of this method in order to define mappings between different UML models can be analyzed.

References

1. K. Baclawski, M. Kokar, P. Kogut, L. Hart, J. Smith, W. Holmes III, J. Letkowski, M. Aronson, Extending UML to Support Ontology Engineering for the Semantic Web, in M. Gogolla, C. Kobryn (Eds.), *UML'01 – The Unified Modeling Language*, LNCS 2185, Springer, 2001, pp. 342–360.
2. K. T. Claypool, E.A. Rundensteiner, G.T. Heineman, Evolving the Software of a Schema Evolution System, in H. Balsters, B. de Brock, S. Conrad (Eds.), *Database Schema Evolution and Meta–Modeling*, LNCS 2065, Springer, 2000, pp. 68–84.
3. E. Domínguez, M.A. Zapata, Mappings and Interoperability: a Meta–Modelling Approach, in T. Yakhno (Eds.), *Adv. in Info. Syst., ADVIS'00*, LNCS 1909, Springer, 2000, pp. 352–362.
4. E. Domínguez, A.L. Rubio, M.A. Zapata, Dynamic Semantics of UML State Machines: a Metamodeling Perspective, *Journal of Database Management*, to be published, 2002.
5. E. Domínguez, M.A. Zapata, J.J. Rubio, A Conceptual approach to meta–modelling, in A. Olivé, J.A. Pastor (Eds.), *Adv. Info. Syst. Eng., CAISE'97*, LNCS 1250, Springer, 1997, pp. 319–332.
6. G. Eckert, P. Golder, Improving object–oriented analysis, *Information and Software Technology*, 36 (2), 1994, pp. 67–86.
7. A. S. Evans, Reasoning with UML class diagrams, *Workshop on Industrial Strength Formal Methods, WIFT'98*, IEEE Press, 1998.
8. D. Firesmith, B. Henderson-Sellers, I. Graham, *The OML Reference Manual*, SIGS Books, 1997.
9. J.C. Grundy, J.R. Venable, Providing Integrated Support for Multiple Development Notations, in J. Iivari, K. Lyytinen (Eds.), *Adv. Info. Syst. Eng., CAISE'95*, LNCS 932, Springer, 1995, pp. 255–268.
10. B. Henderson-Sellers, C. Atkinson, D. Firesmith, Viewing the OML as a variant of the UML, in R. France, B. Rumpe (Eds.), *UML'99 – The Unified Modeling Language*, LNCS 1723, Springer, 1999, pp. 49–66.
11. B. Henderson-Sellers, A. Bulthuis, *Object-Oriented Metamethods*, Springer, 1997.
12. B. Henderson-Sellers, D. Firesmith, I. Graham, OML metamodel: relationships and state modelling, *Journal of Object-Oriented Programming (ROAD)*, SIGS Publications, 10, 1, 1997, pp. 47–51.
13. S. K. Kim, D. Carrington, Formalizing the UML Class Diagram Using Object–Z, in R. France, B. Rumpe (Eds.), *UML'99 – The Unified Modeling Language*, LNCS 1723, Springer, 1999, pp. 83–98.
14. C. Kobryn, Architectural Patterns for Metamodeling, in A. Evans, S. Kent, B. Selic (Eds.), *UML'00 – The Unified Modeling Language*, LNCS 1939, Springer, 2000, p. 497.
15. K. Lano, J. Bicarregui, Semantics and Transformations for UML Models, in J. Bezivin, P.-A. Muller (Eds.), *UML'98 – The Unified Modeling Language*, LNCS 1618, Springer, 1998, pp. 107–119.
16. C. Nicolle, D. Benslimane, K. Yetongnon, Multi–Data models translations in Interoperable Information Systems, in J. Mylopoulos, Y. Vassiliou (Eds.), *Adv. Info. Syst. Eng., CAISE'96*, LNCS 1080, Springer, 1996, pp. 176–192.
17. A. Olivé, M.R. Sancho, Porting ROSES to UML - An Experience Report, in J. Bezivin, P.-A. Muller (Eds.), *UML'98 – The Unified Modeling Language*, LNCS 1618, Springer, 1998, pp. 64–77.
18. OMG, UML Specification Version 1.4 formal/01-09-67. Available at http://www.omg.org. September, 2001.
19. G. Sunyé, D. Pollet, Y. Le Traon, J.M. Jézéquel, Refactoring UML models, in M. Gogolla, D. Kobryn (Eds.) *UML'01 – The Unified Modeling Language*, LNCS 2185, Springer, 2001, pp. 134–148.