Journal of Computer Science 8 (8): 1205-1211, 2012
ISSN 1549-3636
© 2012 Science Publications

# Model Refinement in the Model Driven Architecture Context

[1,2]Luciane Telinski Wiedermann Agner,
[1,2]Inali Wisniewski Soares, [1]Paulo Cezar Stadzisz and [1]Jean Marcelo Simao
[1]Graduate School of Electrical Engineering and Computer Science,
Federal University of Technology Parana (UTFPR),
Av. 7 de Setembro, 3165, 80.230-901, Curitiba, Parana, Brazil
[2]Department of Computer Science,
Faculty of Computer Science, Mid-West State University (UNICENTRO),
Rua Padre Salvador, 875, 85.015-430, Guarapuava, Parana, Brazil

**Abstract: Problem statement:** Model Driven Architecture (MDA) is a software development approach based on the design and the transformation of models. In MDA, models are systematically translated to other models and to a source code. Model transformation plays a key role in MDA. Several model transformation languages have been launched lately, aiming to facilitate the translation of input models to output models. The employment of such languages in practical contexts has succeed, although quite often those languages cannot be directly applied to a particular type of model transformation, called refinement. **Approach:** This study provides a general overview on model refinement and investigates two approaches for model refinement based on Atlas Transformation Language (ATL) referred to as: Refining mode and module superimposition. ATL is a widely adopted language for solving model transformation problems in the MDA approach. **Results:** This study presents the comparative results obtained from the analysis of the Refining Mode and the Module Superimposition approaches, emphasizing their application benefits. **Conclusion:** The increasing use of MDA for the design of software systems empowered researches on how developers may benefit from approaches that perform model refinement. The main advantages achieved with the use of the Module Superimposition technique are maintainability and reusability improvement, obtained through module composition and rule superimposition. In its turn, the Refining Mode stands out for its ease of use.

**Key words:** Model transformation, model refinement, refining mode, module superimposition

## INTRODUCTION

Model Driven Architecture (MDA) promotes the use of models as the main artifacts through all software development stages: System specification, project, implementation and tests (Touzi *et al*., 2009). MDA proposal consists in reducing the semantic distance between the business domain and the implementation platform domain. In order to achieve that, high-level abstraction models focus on protecting software system developers from the complexity of platforms.

In the MDA approach a model is used for generating another model and those models may be either in the same or in different abstraction levels. More abstract models are more distant from the particularities of a software platform, while less abstract models are closer to such specifications. In addition, implementations can fully or partially derive from their models through the application of model transformations (Singh and Sood, 2009). According to the MDA approach, software design comprises the following stages (Touzi *et al*., 2009):

- Specification of a PIM - Platform Independent Model
- Specification of a PM-Platform Model
- Selection of a specific platform for the system
- Transformation of a PIM into a PSM - Platform Specific Model, based on a PM
- Transformation of the PSM into a software system code

**Corresponding Author:** Luciane Telinski Wiedermann Agner, Graduate School of Electrical Engineering and Computer Science, Federal University of Technology Parana (UTFPR), Av. 7 de Setembro, 3165, 80.230-901, Curitiba, Paraná, Brazil Tel: + 55 41 3310 4759

Platform-independent models represent the system functionalities and are developed with the aid of a modeling language, such as Unified Modeling Language (UML). UML is a general purpose modeling language applicable across different domains. Currently, UML is the standard modeling language for software design and, therefore, plays a key role in MDA. A platform can be defined as a set of hardware or software mechanisms that enable the execution of software applications. In its turn, a platform model provides a set of technical concepts that represent components and services of a concrete platform (Dube and Dixit, 2012).

The notion of transformation is an essential issue in the MDA approach. The model transformation scenario are presented in (Dube and Dixit, 2012). Transformation between models can be defined as the translation of a model from a higher abstraction level to a lower abstraction level, based on a set of clearly defined rules (Singh and Sood, 2009). The PIM is transformed into a PSM by means of a model transformation, being the PSM the combination between PIM and the details of a specific implementation platform, by means of the Platform Model (PM).

This study focuses on transformations of PIM models into PSM models (PIM-into-PSM). Within the scope of this study, transformation is a refinement of models that incorporates details of a specific platform to the source model (PIM), being both PIM and PSM models based on the UML metamodel. In a model refinement most elements from the source model (PIM) are copied to the target model (PSM), while other elements must be changed in order to incorporate platform-specific aspects. According to Briand *et al*. (2009) "model-driven development practices rely on the stepwise refinement of analysis models into increasingly detailed design models, all the way down to implementation".

Under this perspective, several languages were proposed so as to define and execute model-to-model (M2M) transformations. One of the most prominent among these languages is called Atlas Transformation Language (ATL) (Tolosa *et al*., 2011). ATL is widely recognized as a solution for the development of model transformations (Troya and Vallecillo, 2011). Two techniques used for model refinement in ATL are explored in this study: Refining Mode and Module Superimposition. Execution modes, structure, benefits and technical limitations are thus described.

## MATERIALS AND METHODS

**Atlas Transformation Language (ATL):** ATL is a model transformation language based on rules established by the Institut National de Recherche en Informatique et en Automatique (INRIA) in response to a request from the Object Management Group (OMG) to propose a model transformation language that is compatible with the QVT standard (Queries/Views/Transformations) (Amstel *et al*., 2011).

In the ATL context, the definition of models is performed according to their metamodels, as presented in Fig. 1. In this way, transformation rules clearly point towards how the source metamodel concepts are mapped in the target metamodel concepts. A transformation from a source Model (Ma) into a target Model (Mb) is therefore conducted by a transformation definition (mma2mmb.atl), based on ATL constructs. In its turn, the transformation definition is also a model (Jouault *et al*., 2008). Then, the source and target models and the transformation definition must conform to their metamodels (MMa, MMb and ATL, respectively). In addition, the metamodels must conform to a meta-metamodel, in this case the Meta Object Facility (MOF) (Dube and Dixit, 2012).

ATL is a hybrid, deeply expressive language that makes use of declarative and imperative constructs. Declarative constructs are clear and accurate, thus more often used for writing transformations. Such constructs allow expressing associations between the source model elements and the target model elements by means of an arrangement of rules. Additionally, imperative constructs enable the simplified specification of complex problems (Tolosa *et al*., 2011).

An ATL transformation is designed according to the following elements: header, import, helpers and transformation rules (Jouault *et al*., 2008). Helpers and rules are constructs used for specifying the transformation functionalities. The term "execution mode" refers to the act of transforming models. There are two execution modes for ATL modules: normal (default) mode and refining mode.

The header section (mandatory) defines the transformation module name and specifies the source and target models. In their turn, such models must be associated with their respective metamodels. Figure 2 brings an example of a transformation module header named PIM2PSM.atl. Such header makes use of the standard execution mode, set through the keyword "from" and defines the PIM as source model (IN). The output model, named OUT, refers to the PSM and is created as a result of the transformation. Both models conform to the UML2 metamodel.

The import section consists of ATL libraries containing a set of general purpose functions, like for example string manipulation functions. An ATL helper is a query based on the Object Constraint Language (OCL), a language used for describing expressions in UML models OMG, 2010.

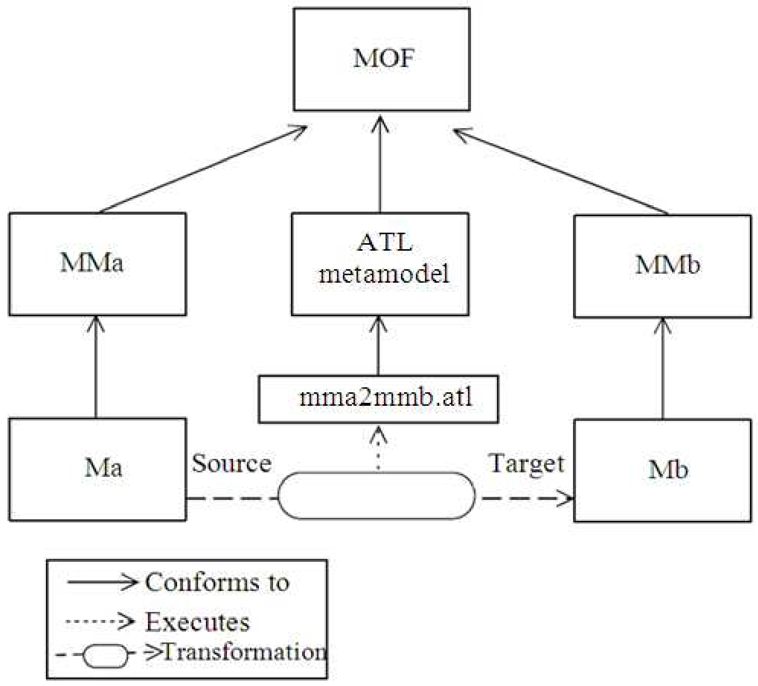

Fig. 1: ATL model transformation standard (Jouault and Kurtev, 2006)

```
module PIM2PSM;
create OUT : UML2 from IN : UML2;
```

Fig. 2: Configuration header

```
module PIM2PSM;
create OUT : UML2 refining IN : UML2;
```

Fig. 3: Refining Mode: Configuration header

Transformation rules are distinguished between matched rules and called rules. Matched rules comply with the declarative approach and are automatically executed. A matched rule specifies a mapping between a set of elements from the source model and a set of elements from the target model. Thus, matched rules are used to implicitly match source elements and produce target elements. As opposed to matched rules, a called rule may take parameters and has to be invoked from an ATL imperative block in order to be executed. Thus, called rules comply with the imperative approach and must be explicitly invoked by another rule.

Also, there is a specific type of matched rule, namely lazy rule that does not automatically trigger. Therefore, a lazy rule is triggered by other rules (Troya and Vallecillo, 2011). The difference between lazy and called rules is that called rules have a parameter specification, whereas lazy rules have a matching specification, like matched rules.

ATL is part of the Eclipse Modeling Framework (EMF), a modeling framework for the design of tools based on structured data models (Amstel *et al*., 2011). In addition, ATL accepts several models as input in the transformation process. ATL transformations are one-way and access the source and target models in the read-only and write-only modes, respectively.

**Model refinement:** This study comprises a PIM-into-PSM model transformation and refers to an endogenous refinement of models based on the UML metamodel. In a model refinement most elements from the source model (PIM) are copied to the target model (PSM), while other elements must be changed in order to incorporate platform-specific aspects. A refinement is a transformation that adds details pertaining to a particular target platform to an existing model (Baudry *et al*., 2010) Performing refinements means transforming an abstract model into a detailed design model, i.e., a top-down evolution.

According to the reference metamodel used to express source and target models, transformations are classified as endogenous or exogenous. In endogenous transformations both source and target models conform to the same metamodel, whereas exogenous transformations occur between models expressed by different metamodels (Sun *et al*., 2009). Because the source (PIM) and the target (PSM) models conform to the same metamodel, the PIM-into-PSM transformation is endogenous.

In ATL it is possible to perform a model refinement by making use of the following approaches: Refining Mode and the composition technique named Module Superimposition. These techniques are detailed next.

**Refining mode:** The Refining Mode is an explicit support for performing ATL refinements in execution mode (Troya and Vallecillo, 2011). ATL has two execution modes, the default execution mode and the refining mode. The Refining Mode is set by adding the keyword "refining" to the transformation module header. Besides, it can be employed only in endogenous transformations, i.e., when both source and target models share the same metamodel. In this manner, elements of the target model are generated by the transformation regarding the type of the elements existing in the source model. All properties of the new elements are, then, started up with the same values defined in the corresponding properties of the source elements. Figure 3 presents the header of a refinement transformation that makes use of the UML2 metamodel as reference for defining the source and target models. To do so, the keyword "refining" must replace the keyword "from" in the transformation header.

The ATL2010 compiler is responsible for implementing the in-place strategy, that is, changes are performed directly in the source model without copying the elements. In so doing, the transformation rules need to specify only the changes to be performed in order to generate the new model, whereas all the other elements remain unchanged. Figure 4 illustrates a Refining Mode transformation that produces a model Ma' from a model Ma based on the in-place strategy. In addition, this version of the ATL compiler supports the deletion of elements, therefore enhancing previous versions (ATL2006 and ATL2004).

**Module superimposition:** Module Superimposition is an internal composition technique in which a transformation module is superimposed by another transformation module (Wagelaar *et al*., 2010). In this way, multiple transformation definitions are combined in a single definition. Consequently, definitions must necessarily use the same model transformation language, e.g., ATL.

The module superimposition technique allows a transformation module to replace certain rules of the superimposed transformation module. The original rule is thus replaced by a new rule with the same name and within the same context. That is to say, the Module Superimposition technique enables the division of the transformation into modules, therefore improving reusability and maintainability of the model transformations.

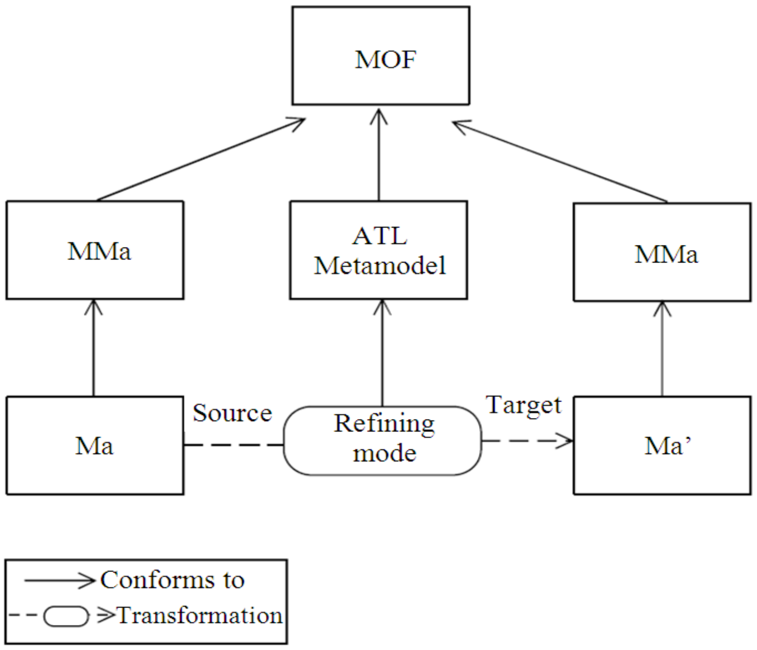

Fig. 4: Refining Mode-in-place strategy

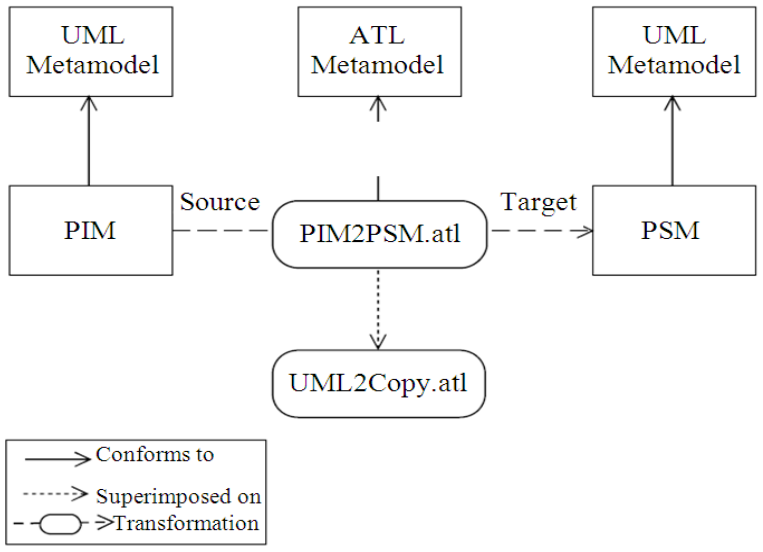

Fig. 5: Module superimposition technique

The UML2Copy.atl module proposed by Wagelaar *et al*. (2010) copies a UML model based on the UML metamodel. Thus, the superimposition technique can make use of this module to solve problems regarding model refinement in the MDA context. In this case, transformation rules of the UML2Copy.atl module are either reused in their original form or, if needed, replaced by homonymous rules defined in the refining specific module. The UML2Copy.atl module consists of approximately 200 rules, including a transformation rule for each metaclass of the UML metamodel.

Figure 5 illustrates a model refinement that makes use of the Module Superimposition technique based on the UML2Copy.atl module. PIM and PSM models stem from the same metamodel, in this case the UML metamodel. The UML2Copy.atl module contains the copying rules of the PIM elements to the PSM. In its turn, module PIM2PSM; the PIM2PSM.atl module contains the refining specific rules that alter the source model (PIM) based on the details of the adopted Platform Model (PM). The transformation modules must comply with the transformation metamodel, being the latter the definition of a Domain-Specific Language (DSL), i.e., the ATL.

## RESULTS

This study proposes a comparative analysis between the refinement approaches presented, i.e., Refining Mode and Module Superimposition using the UML2Copy.atl module. Such analysis assessed the approaches with regard to the following features:

- In-place execution support: Changes are performed directly in the source model without copying the elements to another model
- Apply profiles: Apply the profiles associated with the source model in the target model
- Apply stereotypes: Apply the stereotypes of the source model elements to target model elements.
- Better execution time: Better performance in the transformation execution
- Smaller transformation modules: Number of code lines needed to perform the transformation
- Action blocks support: Support for using imperative code statements so as to set the features of the generated target model element
- Lazy rules support: Support for lazy rules, rules invoked by another rules
- Called rules support: Support for using called rules.
- Complexity: Lower complexity in development and configuration of model transformation
- Iterative target patterns: Makes it possible to generate a set of target model elements conforming to a same type

Table 1 illustrates the support provided by the approaches assessed in comparison to the features analyzed. Refining Mode copies the profiles applied to the source model, as well as the stereotypes applied to the source model elements. On the other hand, in the Superimposition technique the UML2Copy.atl module does not define rules concerning profiles and stereotypes. Such rules must be defined, if needed, thus increasing the complexity of the transformation development process.

Table 1: Support provided by the ATL refinement approaches

| ATL refinement approaches/ Features/Support | Refining Mode` | Module Superimp. |
|---|---|---|
| In-place execution | √ | |
| Apply profiles | √ | |
| Apply stereotypes | √ | |
| Better execution time | √ | |
| Smaller modules | √ | |
| Action blocks support | | √ |
| Lazy rules support | | √ |
| Called rules support | | √ |
| Complexity | √ | |
| Iterative output patterns | | √ |

As depicted by Tisi *et al.* (2011), the transformation execution time is shorter in the Refining Mode in comparison to the Module Superimposition technique. The Refining Mode presented a better performance, once it does not require copying model elements of the source model unchanged part. It is important to point out that transformation time is a relevant aspect to enable an eligible performance of Computer-Aided Software Engineering (CASE) tools, for example. Also, the Refining Mode allows shorter transformation rules, since the copy of unchanged properties and references is not necessary (Tisi *et al*., 2011).

Some advanced features available in the ATL standard mode are not supported by the Refining Mode, for instance: lazy rules, called rules, iterative output patterns and action blocks (Eclipse, 2012). An action block is a sequence of imperative statements that can be used in both matched and called rules. Imperative statements in ATL are the usual constructs for attributing assignments and control flow: conditions and loops. In the development of more complex transformations those advanced features play a key role and the lack of them often hinders the development of such transformations.

Besides the advantages regarding better performance and shorter code size, programming refinement transformations in the Refining Mode is simpler and easier, once it dispenses with both the expertise in the UML2Copy.atl module and the advanced configuration of the module superimposition. On the contrary, the Refining Mode has limitations that often hamper the development of more complex transformations, e.g., transformations that define imperative constructs. On the other hand, Module Superimposition can deal with nonstandard situations, such as defining imperative statements (Wagelaar *et al*., 2010).

## DISCUSSION

The results obtained demonstrated that the Refining Mode is simpler to use and had a better performance when compared to the Module Superimposition technique. Refining Mode does not support advanced features needed for the design of more complex transformations, e.g., the ones involving lazy and called rules.

The employment of the Module Superimposition technique for UML-based refinements requires the use of the UML2Copy.atl module. That is to say, it is necessary that the developer masters the rules defined in this module so as to superimpose these rules according to the specific transformation requirements. Another aspect taken into account was that the stereotypes annotated in the PIM elements were not copied to the PSM, once the UML2Copy.atl module does not define any rule for copying the profile application and the stereotypes existing in the source model.

As acknowledged by the authors, the techniques presented in this study are the most spread and adopted for ATL model refinement. Other researches explore the model refining implementation in other model transformation languages (Kapova and Goldschmidt, 2009; Guerra *et al*., 2011). Those researches are not oriented to the ATL language, nevertheless. Tisi *et al*. (2011) dealt with model refinement by using rule-based languages, such as ATL. However, this study assessed the Refining Mode and Module Superimposition approaches only with regard to the following aspects: Execution performance and code final size. Therefore, this research did not depict all criteria hereby assessed.

## CONCLUSION

The aim of the study was to evaluate the existing techniques used in ATL model refinement. The main advantages pointed out by the Refining Mode are: Use straightforwardness and execution quickness. On the other hand, this technique has severe restrictions, such as: incompatibility with action blocks and lazy rules. These restrictions hamper and quite often hinder the development of more complex model transformations.

Composition techniques are considerably new in the domain of model transformation languages. This study assessed the composition technique named Module Superimposition using UML2Copy.atl. The main advantages obtained with the use of this technique are: maintainability and usability improvement, obtained through module composition and rule superimposition in the same context. Further, this technique proved to be more flexible and efficient, once it got rid of the limitations present in the Refining Mode. However, it is worth pointing out that the Module Superimposition technique requires the developer’s mastery with regard to the configuration of ATL composition techniques and to the UML2Copy.atl module. In addition, the UML2Copy.atl module neither defines the rules for copying a profile application to a model, nor specifies the rules that apply the stereotypes existing in the source model to the target model.

Therefore, the choice between one of the techniques presented must be pondered as the case may be, relying on the features and transformation requisites intended to be developed.

## REFERENCES

Amstel, M.V., S. Bosems, I. Kurtev and L.F. Pires, 2011. Performance in model transformations: Experiments with ATL and QVT. Theory Pract. Model Transform., 6707: 198-212. DOI: 10.1007/978-3-642-21732-6_14

Baudry, B., S. Ghosh, F. Fleurey, R. France and Y.L. Traon *et al*., 2010. Barriers to systematic model transformation testing. Mag. Commun. ACM, 53: 139-143. DOI: 10.1145/1743546.1743583

Briand, L.C., Y. Labiche and T. Yue, 2009. Automated traceability analysis for UML model refinements. Inform. Software Technol., 51: 512-527. DOI: 10.1016/j.infsof.2008.06.002

Dube, M.R. and S.K. Dixit, 2012. Modeling theories and model transformation scenario for complex system development. Int. J. Comput. Appli., 38: 11-18.

Eclipse, 2012. ATL/User Guide-The ATL Language. The Eclipse Foundation.

Guerra, E., J.D. Lara, D.S. Kolovos, R.F. Paige and O.M.D. Santos, 2011. Engineering model transformations with *transML*. Software Syst. Model. DOI: 10.1007/s10270-011-0211-2

Jouault, F. and I. Kurtev, 2006. Transforming models with ATL. Satellite Events MoDELS 2005 Conf., 3844: 128-138. DOI: 10.1007/11663430_14

Jouault, F., F. Allilaire, J. Bezivin and I. Kurtev, 2008. ATL: A model transformation tool. Sci. Comput. Programm., 72: 31-39. DOI: 10.1016/j.scico.2007.08.002

Kapova, L. and T. Goldschmidt, 2009. Automated feature model-based generation of refinement transformations. Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications, Aug. 27-29, IEEE Xplore Press, Patras, pp: 141-148. DOI: 10.1109/SEAA.2009.67

Singh, Y. and M. Sood, 2009. Model driven architecture: A perspective. Proceedings of the IEEE International Advance Computing Conference, Mar. 6-7, IEEE Xplore Press, Patiala, pp: 1644-1652. DOI: 10.1109/IADCC.2009.4809264

Sun, Y., J. White and J. Gray, 2009. Model transformation by demonstration. Model Driven Eng. Languages Syst., 5795: 712-726. DOI: 10.1007/978-3-642-04425-0_58

Tisi, M., S. Martinez, F. Jouault and J. Cabot, 2011. Refining models with rule-based model transformations. Institut National De Recherche En Informatique Et En Automatique.

Tolosa, J.B., O. Sanjuan-Martinez, V. Garcia-Diaz, B.C.P. G-Bustelo and J.M.C. Lovelle, 2011. Towards the systematic measurement of ATL transformation models. Software: Pract. Exp., 41: 789-815. DOI: 10.1002/spe.1033

Touzi, J., F. Benaben, H. Pingaud and J.P. Lorre, 2009. A model-driven approach for collaborative service-oriented architecture design. Int. J. Produ. Econ., 121: 5-20. DOI: 10.1016/j.ijpe.2008.09.019

Troya, J. and A. Vallecillo, 2011. A rewriting logic semantics for ATL. J. Object Technol., 10: 1-29. DOI: 10.5381/jot.2011.10.1.a5

Wagelaar, D., R.V.D. Straeten and D. Deridder, 2010. Module superimposition: A composition technique for rule-based model transformation languages. Software Syst. Model., 9: 285-309. DOI: 10.1007/s10270-009-0134-3