A Case Study Based Comparison of ATL and SDM

Sven Patzina and Lars Patzina

Center for Advanced Security Research Darmstadt (CASED), Germany
{sven.patzina,lars.patzina}@cased.de

Abstract. In model driven engineering (MDE) model-to-model transformations play an important role. Nowadays, many model transformation languages for different purposes and with different formal foundations have emerged. In this paper, we present a case study that compares the Atlas Transformation Language (ATL) with Story Driven Modeling (SDM) by focusing on a complex transformation in the security domain. Additionally, we highlight the differences and shortcomings revealed by this case study and propose concepts that are missing in both languages.

Keywords: Atlas Transformation Language, Story Driven Modeling, Live Sequence Charts, Monitor Petri nets, transformation.

1 Introduction

Model-driven engineering (MDE) demands model-to-model transformations between models on different abstraction levels. Based on this idea, a model-based development process for security monitors [15] is developed that allows for the abstract specification and automated generation of correct security monitors in software (C, Java) and hardware (VHDL, Verilog). Specifications are modeled as use and misuse cases with extended Live Sequence Charts (LSCs). Due to the expressiveness of LSCs, the process foresees a more explicit intermediate language – the Monitor Petri nets (MPNs), a Petri net dialect with special start and end places and deterministic execution semantics. This more explicit representation with a less complex syntax is easier to process than the LSCs itself.

In this context, a rule-based model-to-model transformation language is intended for the complex step from LSCs to MPNs, because a rule-based approach seems to be less error-prone compared to a manual implementation of the pattern matching process for each transformation in a general-purpose programming language. Nowadays, various transformation languages have emerged with a different purpose, feature set, and formal foundation. On one side, there are languages that are based on graph grammar theory such as SDM [6] and on the other side, languages such as ATL [9], whose semantics has been formalized by using e.g., abstract state machines [5] and rewriting logics [18]. In contrast to existing comparisons that use classical examples [4] or more complex examples by focusing on special properties such as inheritance [21], this case study differs in the application domain.

A. Schürr, D. Varró, and G. Varró (Eds.): AGTIVE 2011, LNCS 7233, pp. 210–221, 2012.
© Springer-Verlag Berlin Heidelberg 2012

In this paper, we show the differences, advantages and disadvantages of the Atlas Transformation Language (ATL) in version 3.1 and Story Driven Modeling (SDM) on an example of a real-world transformation. Our main contributions are the application of two transformation languages to a complex, real-world example in the security domain and specific proposals for extending the concepts of the transformation languages.

In the following, Sec. 2 introduces the transformation scenario. Then Sec. 3 presents the requirements and analyses appropriate languages for our purpose – ATL and SDM. In Sec. 4, selected rules of the transformation in ATL and SDM that show concepts and differences are compared, and the evaluation is described. The result of the comparison and suggestions for additional features for the two transformation languages are shown in Sec. 5. Section 6 concludes the paper.

2 Running Example

In this section, a Car-to-Infrastucture scenario, where a `car` communicates with a `tollbridge`, will be presented and used in the remainder of this paper. The example is based on metamodels that are reduced versions of those used in the case study, depicted as Ecore/MOF diagrams in Figure 1a) and b). Figure 1c) shows a communication protocol (use case) in concrete LSC syntax and Figure 1d) the corresponding MPN representation.

LSCs are an extension of Message Sequence Charts that in addition offer a distinction between *hot* and *cold* elements. Thereby, hot elements are mandatory and have to occur and cold elements are optional. Additionally, LSCs can have two forms, a *universal LSC* with a *prechart* (precondition) before the *mainchart* or an *existential LSC* without a precondition.

The LSC in Figure 1c) shows an existential LSC that models the exchange of asynchronous messages between LSC objects in a mainchart. When the car approaches the tollbridge, it sends a `connect()` message to the tollbridge that is represented as hot message. The tollbridge acknowledges this message with an `ack()`. After receiving this message, the car has to send some information `data_a()`. Then the car is allowed to send additional information `data_b()`, modeled as cold message. The communication has to be terminated by the car by sending a `disconnect()` message.

The metamodel of the *LSC* diagrams is depicted in Fig. 1a), where *LSC*s and the derived *ExistentialLSC*s are contained in the root class *LSCDocument*. Furthermore, *ExistentialLSC*s contain *LSCObject*s and a *Mainchart*. A *Message* starts and ends on a *Location* that is contained in the related *LSCObject* as an ordered set.

Similar to *LSC*s, *MPN*s have an *MPNDocument* as root class, depicted in Figure 1b). The MPN, derived from the use case LSC, is represented in Figure 1d). This MPN is a more operational description of the behavior expressed by the LSC. *MPN*s are composed of different kinds of places (depicted as circles), transitions (realized as black bars), and arcs that connect places and transitions.

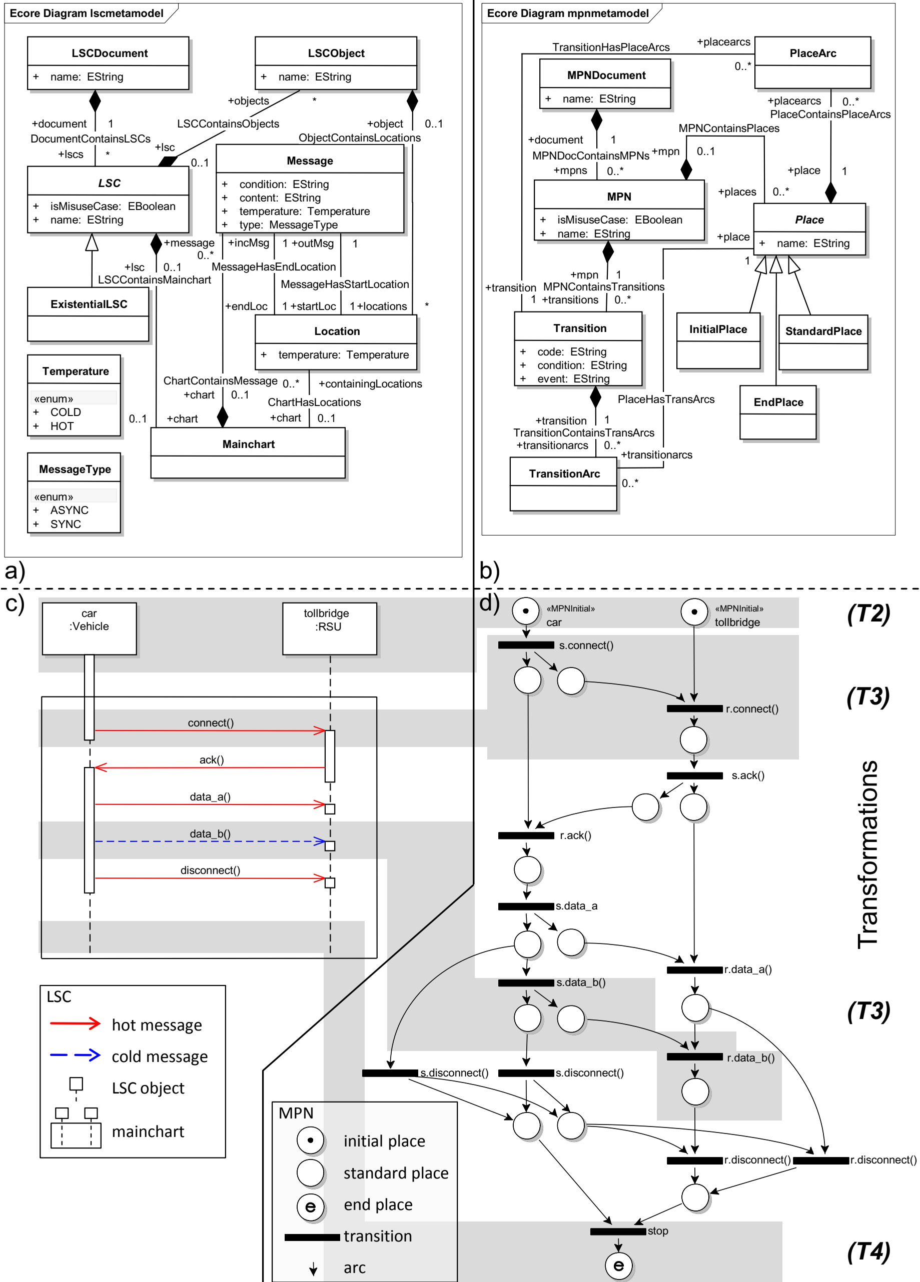


Fig. 1. Metamodels and models of the LSC-to-MPN example

The transformation from LSCs to MPNs basically consists of four transformation steps, highlighted in grey in the concrete example. *(T1)* It starts with the creation of an MPN for each LSC. *(T2)* Each *LSC object* is represented as an *initial place* annotated with the name of the object. *(T3)* A *hot asynchronous message* such as the `connect()` message is split in one *transition* for the sending and one for the receiving *event* and three *standard places*. These are connected with *arcs* where one place is on the sender side, one is on the receiver side, and one secures the order of sending and receiving of the message. This pattern is also used for *cold messages* with additional bypass-transitions that realize the optional nature of cold messages. *(T4)* The pattern is finalized by an *end place* where all possible paths through the MPN are synchronized by transitions with the *event* “stop”.

3 Related Work

In the last years, publications about comparisons between different transformation languages evolved. On the one side, there is the Transformation Tool Contest[1] event series, where solutions for special transformation problems are submitted and compared. As [2] stated, this contest can be used as source for insights of the strong and weak points of transformation tools, but has no clear focus on achieving really comparable results. On the other side, publications cope with small classical examples such as UML2RDBMS [4], concentrate only on graph-based transformation languages [17] or focus on a small subset of properties such as inheritance [21]. The transformation from extended LSCs to the corresponding MPN representation is in contrast to the afore mentioned comparisons based on a complex, real-world example located in the security domain.

Based on the transformation steps (*T1* to *T4*) that are derived from the example in Section 2 the following requirements can be formulated:

R1) 1-to-1 (Model). For each LSC diagram, a single MPN should be generated. Therefore, for each source model one target model is created. *(T1)*
R2) m-to-n (Element). One or more elements in the source model have to be mapped to one or more elements in the target model. *(T3)*
R3) Traceability Links. For processing optional elements, traceability links are needed to be able to add bypass-transitions in the target model. *(T3)*
R4) Attributes. The language must be able to handle attributes of model elements. It has to check and generate attributes in the target model. *(T2)*
R5) In-place. For optimizations of the target model, some kind of in-place transformation on the target model is required. *(T4)*
R6) Deletion. For post-processing, it is necessary to delete elements from the target model to remove redundant places and unnecessary transitions. *(T4)*
R7) Recursive Rules. For the synchronization at the end of an LSC, with an unknown number of places and LSC objects during specification, recursive operations on model elements are needed. *(T4)*

[1] `http://planet-research20.org/ttc2011/`

Table 1. Requirements for the LSC-to-MPN transformation

Req.	ATL [9]	ETL [11]	QVTo [14]	PROGRES [16]	SDM [6]	TGG [10]	VIATRA2 [20]
R1)	✓	✓	✓	✓[1]	✓[1]	✓	✓
R2)	✓	–[2]	✓	✓	✓	✓	✓
R3)	✓	o[3]	✓	o[3]	o[3]	✓	o[3]
R4)	✓	✓	✓	✓	✓	–	✓
R5)	o[4]	✓	o[4]	✓	✓	–	✓
R6)	o[5]	✓[6]	✓	✓	✓	–	✓
R7)	o[7]	✓	✓	✓[8]	✓[8]	–	✓
R8)	✓	✓	✓	✓	✓	o[9]	✓
R9)	✓	✓	✓	–	✓	✓	✓

✓fulfilled; o partly fulfilled; - not fulfilled; [1]in-place; [2]only 1-to-n; [3]manual; [4]refining mode; [5]new transformation; [6]with EOL; [7]as helper; [8]control flow and path expressions; [9]bidirectional

R8) Unidirectional. Some elements of the LSC have no bijective mapping. A fixed loop of n iterations is, e.g., unwound to n representations of its content. *(T1)*

R9) Tool Support. For the realization of the development process, a reliable implementation of the transformation language is needed.

Based on these requirements, Table 1 compares state-of-the-art rule-based model transformation languages. While PROGRES, SDM, TGG, and VIATRA2 are based on graph transformation (GT) principles, ATL, ETL, and Operational QVT (QVTo) are not formalized. First approaches for ATL are made using abstract state machines [5] and rewriting logics [18]. The introduced GT languages, excluding TGGs, are hybrid and support the modeling of control flow. In contrast to all other approaches, TGGs are fully declarative and bidirectional transformations have to be specified as mappings of source and target elements simultaneously. This hampers the specification of the rules, because no bijective mapping as stated in R8 exists, and R7 is not supported. So TGG does not fit to the requirements.

There are two groups, on one hand ATL, ETL, and QVTo and on the other hand PROGRES, SDM, and VIATRA2. Because many comparisons between languages within one of the groups exist, e.g., [8,17], one language from each group is chosen.

ATL, the commonly used model transformation language in the Eclipse community and SDM that is used in the meta-CASE tool MOFLON [1] will be compared because of their differences. In SDM the rules, embedded in activities of an activity diagram, are scheduled by an explicitly modeled control flow. Contrary to SDM, the rules of an ATL 3.1 transformation are conditioned by OCL expressions and functionality can be delegated to helper functions. Using these concepts the execution sequence is derived from implicit relations between the rules.

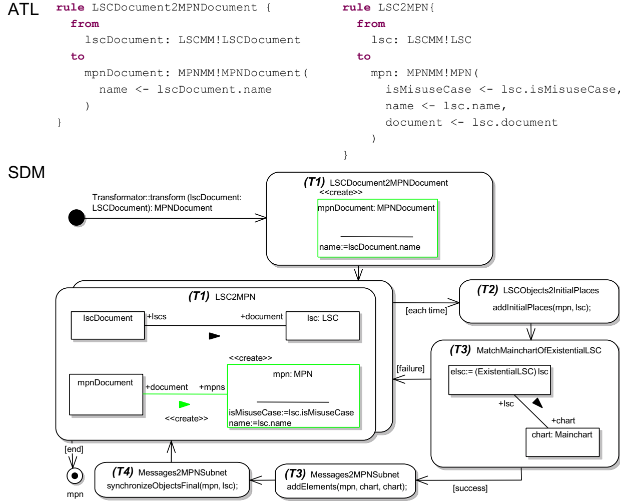


Fig. 2. Initial rules of the transformation (T1)

4 Comparison of the Transformations

In this section, rules of the LSC-to-MPN transformation are presented that show commonalities and differences of ATL and SDM. Thereby, the requirements derived above that are not satisfied by one or both languages (R3, R5, R6 and R7) are examined and additional missing features are suggested. After that, the implementation of the transformations is evaluated.

4.1 Transformations

The SDM part of Figure 2 maps the basic steps of the transformation, derived in Section 2 to the activities of the SDM: *(T1) LSCDocument* to *MPNDocument* and *LSC* to *MPN*, *(T2) LSCObjects* to *InitialPlaces*, *(T3) Messages* to an *MPN* subnet, and *(T4)* the synchronization to *EndPlaces* in *MPN*.

(T1) The first transformation rules in Figure 2 translate the *LSCDocument* with its *LSC*s to an *MPNDocument* with corresponding *MPN*s. The ATL rule, *LSCDocument2MPNDocument*, corresponds to the first activity in the SDM. In the SDM rule, *lscDocument* is already bound as a parameter and an *MPNDocument* is created in the first activity. Contrary to the SDM, the ATL rule needs no already matched (bound) objects for the navigation. So an *LSCDocument* is matched in the *from* part (left side) and an *MPNDocument* is created in the *to* part (right side) of the rule. The second rule *LSC2MPN* is very similar in both languages. In the SDM, the activity around the pattern is a *foreach*-activity that uses the bound *lscDocument* to find all *LSC*s and adds for each *lsc* an MPN with the same attributes to the *mpnDocument*.

These first transformation rules already reveal the main difference of the two languages. While SDM explicitly relies on a control flow between the declarative

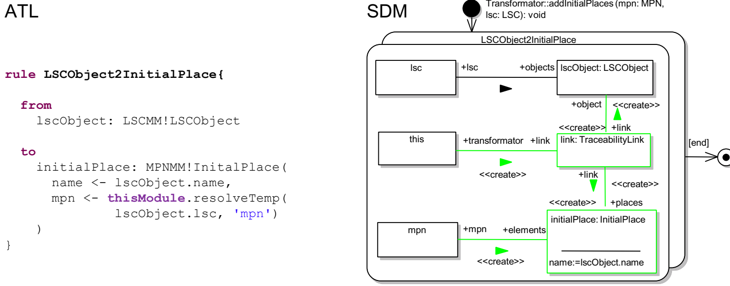


Fig. 3. Creation of initial places of MPN (T2)

patterns, ATL should be used as long as possible in a declarative way [9]. Beside the implementation of SDM in MOFLON, there exists a backward compatible extension that allows for implicit rule scheduling [13].

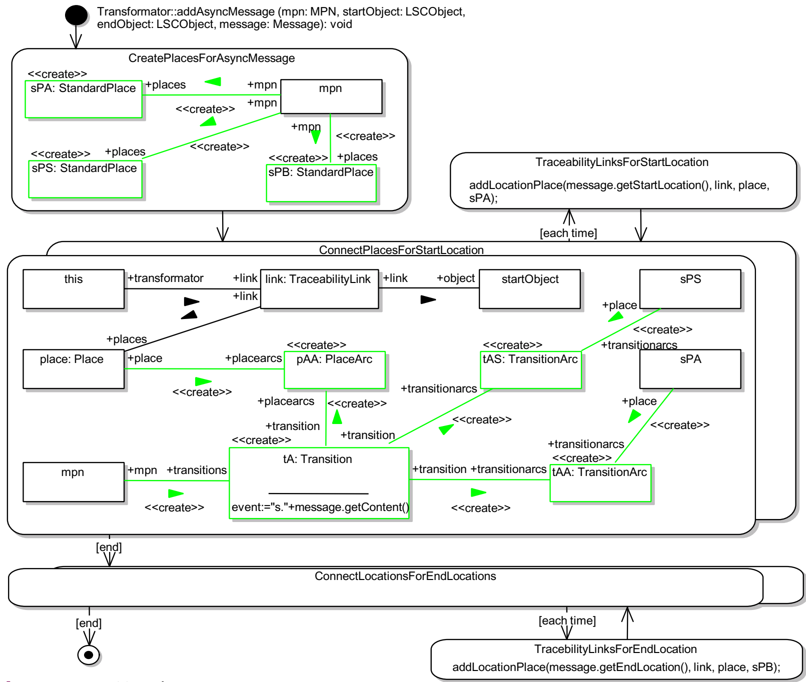
(T2) For every matched LSC in the activity *LSC2MPN*, the transformation steps are executed. The next activity calls the rule depicted in Figure 3 that translates all *LSCObject*s to *InitialPlace*s of the MPN. The ATL rule matches every *LSCObject* specified in the *from* part of the rule, which corresponds to the unbound object *lscObject* in the SDM. The *to* part of the ATL rule corresponds to the pattern at the bottom of the SDM activity.

Here, another difference between ATL and SDM emerges. In ATL each application of a rule automatically produces traceability links between the matched source and the created target elements. Such a mechanism does not exist in standard SDM. Therefore, an additional metamodel containing the transformation rules as operations and constructs for the management of traceability links (*TraceabilityLink*) has to be explicitly modeled as shown in [7]. Traceability links in ATL have to be used for adding a reference to the already created MPN. This is done by the predefined helper `resolveTemp` and name matching. As parameters the elements of the source model and the name of the element in the target model, defined in the rule that has matched the source elements, are passed.

(T3) When defining more complex rules such as the transformation of asynchronous messages, two different approaches have to be used. In SDM, enabled by the explicit control flow modeling, the transformation can be defined in one rule, whereas, in ATL three rules have to be specified.

For the SDM in Figure 4, the manually added traceability links are used to identify all places in the MPN that have not been synchronized via a hot message in the LSC. In the activity *CreatePlacesForAsyncMessage* three *StandardPlace*s for the LSC message are created. The following *foreach*-activity *CreatePlaces-ForStartLocation* generates a transition with corresponding arcs for every place that has a link to the source of the message (*LSCObject*). This includes the bypass-transitions for cold messages. The *statement*-activity *TraceabilityLink-ForStartLocation* calls an SDM method that manages the traceability links. The second part of the SDM, which is collapsed, performs the similar transformation for the target of the message.

SDM



ATL

```
rule AsyncMPNPattern{
  from
   lscAM : LSCMM!Message(lscAM.isMessageAsync)
  to
    sPA : MPNMM!StandardPlace(
      mpn <- thisModule.resolveTemp(lscAM.endLocation.object.lsc, 'mpn') ),
    sPS : MPNMM!StandardPlace( placeArcs <- tAS,
      mpn <- thisModule.resolveTemp(lscAM.endLocation.object.lsc, 'mpn') ),
    sPB : MPNMM!StandardPlace( placeArcs <- tAB,
      mpn <- thisModule.resolveTemp(lscAM.endLocation.object.lsc, 'mpn') ),
    tA : MPNMM!Transition( transitionArcs <- tAA, transitionArcs <- tAS,
      event <- 's.'.concat(lscAM.content),
      mpn <- thisModule.resolveTemp(lscAM.endLocation.object.lsc, 'mpn') ),
    tAA : MPNMM!TransitionArc( place <- sPA ),
    tAS : MPNMM!TransitionArc( place <- sPS ),
    pAA : MPNMM!PlaceArc( transition <- tA, place <- thisModule.getPlaceOfPrevMsgSender(lscAM) ),
    ...
}
rule bypassTransition{
  from
    firstLoc : LSCMM!Location,
    secondLoc : LSCMM!Location(firstLoc.isBypassCombination(secondLoc))
  to
    pA : MPNMM!PlaceArc( transition <- t,
      place <- secondLoc.getPlaceForLocAndInst ),
    t : MPNMM!Transition( event <- secondLoc.getEventName, transitionArcs <- tA,
      mpn <- thisModule.resolveTemp(firstLoc.object.lsc, 'mpn') ),
    tA : MPNMM!TransitionArc( place <- thisModule.getPlaceForLocAndInst(secondLoc) )
}
rule bypassTransitionWithFollowingAsyncSend extends bypassTransition{
  from
    firstLoc : LSCMM!Location,
    secondLoc : LSCMM!Location(firstLoc.isBypassCombination(secondLoc)
      and secondLoc.isAsyncSendingLocation)
  to
    tAS : MPNMM!TransitionArc( transition <- t,
      place <- thisModule.resolveTemp(secondLoc.outgoingMessage, 'sPS')
    ),
    pAS : MPNMM!PlaceArc ( place <- thisModule.resolveTemp(secondLoc.outgoingMessage, 'sPS'),
      transition <- thisModule.resolveTemp(
        Tuple{fL = firstLoc.getOppositeLocation, sL = secondLoc.getOppositeLocation}, 't') )
}
```

Fig. 4. SDM and ATL rules for asynchronous messages (T3)

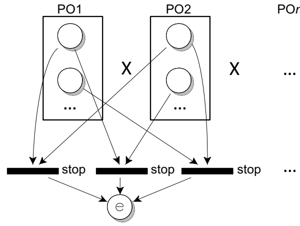


Fig. 5. Synchronization of open places at the end of an MPN (T4)

In ATL the transformation, depicted in Figure 4, is split into three rules that match different source elements. Where *AsyncMPNPattern* matches all messages in the *LSC* model and translates the messages, the two *bypassTransition* rules create the additional transitions for cold messages. For that, the messages have to be matched again. This can be done by splitting the transformation in two sequential transformations, losing the traceability links of the former steps, or as done here, by matching other source elements (combination of locations). The *to* parts of the ATL rules and their helpers extensively use traceability links.
(T4) The LSC presented in Section 2 ends with a hot message, so that only the places belonging to the last message have to be synchronized via a transition with `stop` event. For handling one or more cold messages at the end of a chart, all combinations of places that have not been synchronized from different LSC objects have to be connected to an own transition. As shown in Figure 5, the Cartesian product $(po1, po2, \ldots) \in PO1 \times PO2 \times \ldots$ has to be derived, where POn contains all unsynchronized places of the n-th *LSCObject*.

Such a pattern with two variable dimensions – the number of *LSCObjects* and the number of *Places* for each object – cannot be described in a declarative manner in ATL and SDM. Therefore, a recursive traversal is needed that can only be realized imperatively. In SDM, this can be handled locally in the control flow by calling rules recursively, whereas, in ATL this has to be implemented fully imperatively in global helpers.

4.2 Evaluation of the Transformations

To validate the semantical equivalence of the two transformations, both approaches have been fed with the same set of input models, and the output models have been compared manually with each other. The input models with up to five messages have been chosen to cover different sequences of messages based on the temperature, the type, and the direction between the two LSC objects. Caused by the manual review of the output models, these input models are as compact as possible to realize all combinations with respect to the specifications of the transformations.

To evaluate the equivalence for more complex models, an "automatic" comparison[2] of the output MPNs is needed. Therefore, the part of the SDM transformation that is presented in this paper has been transferred to the eMoflon

[2] Test suite provided at: `http://www.moflon.org/emoflon`

tool, a new version of MOFLON based on EMF. Hereby, EMFCompare allows for an automatic comparison of the output models (MPNs).

5 Desirable Features

In this section, proposals for additional features of these two languages are made. These originate from ***(R)*** the basic requirements of Section 3, ***(T)*** the transformation in Section 4, and ***(S)*** additional requirements for the monitor generation process scenario.

Implicit Traceability Links (R3, T2, T3). The explicit modeling of traceability links in SDM alloy the readability of the transformation rules caused by the additional patterns to create these links. An implicit generation as realized for subgraph copying [19] would be desirable for the here studied more general case.

In-Place Transformation (R5, R6). For post-processing purposes in the MPN target model SDM, as an in-place language, is favorable, because recursive deletions and modifications can be modeled within the control flow. Using the refining mode of ATL 3.1 (realized by copy rules) allows a kind of in-place transformation, but this approach fails when post-processing steps have to be repeated iteratively on the changed model until there is no new match. This is caused by the write-only target model that has to be used as source model in a repeated external call of the transformation. In the implementation of ATL 3.2 an extended support for in-place transformations and explicit deletions has been added, but with the drawback that some advanced imperative features are not supported.

Patterns of Dynamic Size (R7, T4). In complex transformations some problems occur that are typically resolved in programming languages with a recursive approach, which is needed to compute the Cartesian product of an unknown number of sets each containing an unknown number of elements. Figure 5 shows this using the example of the synchronization of all places at the end of the transformation, presented in Section 4.1. To eliminate the imperative part for this issue, a template concept for patterns is needed that allows the dynamic runtime initialization of parts of patterns by a quantity of instances in the model.

Explicit Modeling of Control Flow (T1, T3). When a complex transformation should be described in ATL, every object can only be bound by one rule, which leads to a shortage of unbound elements for other rules, as presented in Section 4.1 in *T3.* This forces the developer to design more complex holistic rules or split the complete transformation into independent sequential steps. By splitting the transformation, traceability links created in a previous step are not accessible in the current step.

Reusability of Matched Patterns (T2). In SDM, set patterns can be used to match many objects of the same type at once. The results can be passed between rules and returned as result but cannot be used for further pattern matching. By extending the set concept and allowing, additionally, the passing of matched patterns, the control flows in SDM rules could be reduced.

Deterministic and Correct Result (S). In the presented monitor generation process the correctness of the resulting model has to be ensured. Therefore, properties such as a deterministic generation of target models, as guaranteed by the declarative part of ATL [8], are desirable. As shown in the previous section, it is often impossible to provide a purely declarative solution for complex transformations. Hence, in both ATL and SDM the developer has to cope with non-determinism in the modeled transformation. To address this issue, test practices, as suggested in [3], have to be developed for ATL and SDM.

Integration into Software (S). The tools for ATL and SDM provide different approaches for the integration of transformations in software products. MOFLON and FUJABA use SDM specifications to generate Java code and ATL is translated into byte-code that is interpreted by a special virtual machine (ATL VM). Hence, the SDM code is preferable for seamless integration in a standalone tool [6]. When developing a tool integrated in Eclipse, both approaches are suitable.

6 Conclusion and Future Work

In this paper, we have presented a case study about the comparison of the transformation languages ATL and SDM in the context of a model-based security monitor development process. We have highlighted shortcomings that have evolved during the case study and suggested additional concepts to improve the modeling of transformations. Both languages lack some features and should be extended. One major disadvantage of ATL is the missing possibility to explicitly model the control flow, and the resulting problem that elements can be bound only once in a transformation.

A more satisfactory model transformation language for our monitor generation process could be based on an SDM-like hybrid language that is extended by static type and determinism analysis from PROGRES and Critical Pair Analysis from AGG [12]. The language should support recursive patterns as implemented in VIATRA. Additionally, a possibility for implicit traceability links should be supported. Furthermore, an improved parameter handling for passing matched patterns and set patterns between part rules is also desirable.

As stated, all these concepts have been proposed for different transformation languages, but were never combined in an implementation of a graph transformation language. Therefore, further research has to determine the compatibility of these extensions, e.g., determinism and recursive patters.

References

1. Amelunxen, C., Königs, A., Rötschke, T., Schürr, A.: MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations. In: Rensink, A., Warmer, J. (eds.) ECMDA-FA 2006. LNCS, vol. 4066, pp. 361–375. Springer, Heidelberg (2006)
2. van Amstel, M., Bosems, S., Kurtev, I., Ferreira Pires, L.: Performance in Model Transformations: Experiments with ATL and QVT. In: Cabot, J., Visser, E. (eds.) ICMT 2011. LNCS, vol. 6707, pp. 198–212. Springer, Heidelberg (2011)

3. Baudry, B., Dinh-Trong, T., Mottu, J.M., Simmonds, D., France, R., Ghosh, S., Fleurey, F., Le Traon, Y.: Model transformation testing challenges. In: ECMDA Workshop on Integration of MDD and MDT. IRB Verlag (2006)
4. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal 45, 621–645 (2006)
5. Di Ruscio, D., Jouault, F., Kurtev, I., Bézivin, J., Pierantonio, A.: Extending AMMA for supporting dynamic semantics specifications of DSLs. Tech. rep. LINA (2006)
6. Fischer, T., Niere, J., Torunski, L., Zündorf, A.: Story Diagrams: A New Graph Rewrite Language Based on the Unified Modeling Language and Java. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) TAGT 1998. LNCS, vol. 1764, pp. 296–309. Springer, Heidelberg (2000)
7. Hildebrandt, S., Wätzoldt, S., Giese, H.: Executing graph transformations with the MDELab story diagram interpreter. In: Transformation Tool Contest (2011)
8. Jouault, F., Kurtev, I.: Transforming Models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)
9. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Science of Computer Programming 72(1-2), 31–39 (2008)
10. Klar, F., Rose, S., Schürr, A.: TiE – a tool integration environment. In: Proc. of the 5th ECMDA-TW. CTIT Workshop Proc., vol. WP09-09, pp. 39–48 (2009)
11. Kolovos, D.S., Paige, R.F., Polack, F.A.C.: The Epsilon Transformation Language. In: Vallecillo, A., Gray, J., Pierantonio, A. (eds.) ICMT 2008. LNCS, vol. 5063, pp. 46–60. Springer, Heidelberg (2008)
12. Mens, T., Taentzer, G., Runge, O.: Detecting structural refactoring conflicts using critical pair analysis. In: Proc. of the Workshop on Software Evolution through Transformations. ENTCS, vol. 127, pp. 113–128. Elsevier (2005)
13. Meyers, B., Van Gorp, P.: Towards a hybrid transformation language: Implicit and explicit rule scheduling in story diagrams. In: Proc. of the 6th Int. Fujaba Days (2008)
14. OMG: MOF 2.0 QVT Spec. Object Management Group (January 2011), http://www.omg.org/spec/QVT/1.1/
15. Patzina, S., Patzina, L., Schürr, A.: Extending LSCs for Behavioral Signature Modeling. In: Camenisch, J., Fischer-Hübner, S., Murayama, Y., Portmann, A., Rieder, C. (eds.) SEC 2011. IFIP AICT, vol. 354, pp. 293–304. Springer, Heidelberg (2011)
16. Schürr, A.: Programmed Graph Replacement Systems. In: Handbook of Graph Grammars and Computing by Graph Transformation, vol. 1: Foundations, pp. 479–546. World Scientific (1997)
17. Taentzer, G., Ehrig, K., Guerra, E., de Lara, J., Lengyel, L., Levendovszky, T., Prange, U., Varró, D.: Varró-Gyapay, Sz.: Model transformation by graph transformation: A comparative study. In: Proc. of Workshop MTiP (2005)
18. Troya, J., Vallecillo, A.: Towards a Rewriting Logic Semantics for ATL. In: Tratt, L., Gogolla, M. (eds.) ICMT 2010. LNCS, vol. 6142, pp. 230–244. Springer, Heidelberg (2010)
19. Van Gorp, P., Schippers, H., Janssens, D.: Copying subgraphs within model repositories. In: Proc. of the 5th Int. Workshop on Graph Transformation and Visual Modeling Techniques. ENTCS, vol. 211, pp. 133–145. Elsevier (2008)
20. Varró, D., Balogh, A.: The model transformation language of the VIATRA2 framework. Science of Computer Programming 68(3), 214–234 (2007)
21. Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W., Schönböck, J., Schwinger, W., Kolovos, D., Paige, R., Lauder, M., Schürr, A., Wagelaar, D.: A Comparison of Rule Inheritance in Model-to-Model Transformation Languages. In: Cabot, J., Visser, E. (eds.) ICMT 2011. LNCS, vol. 6707, pp. 31–46. Springer, Heidelberg (2011)