Comparing Two Model Transformation Approaches

Jochen M. Küster and Shane Sendall and Michael Wahler

Computer Science Department
IBM Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
email: [jku,sse,wah]@zurich.ibm.com

Abstract. For the MDA vision to become a reality, there must be a viable means to perform model-to-model transformation. In this paper, we compare and contrast two approaches to model transformation: one is a graph transformation-based approach, and the other is a relational approach, based on the QVT-Merge submission for OMG's MOF 2.0 Query/View/Transformation Request for Proposal. We apply them both to a common example, which involves transforming UML statemachines to a CSP specification, and we look at some of the concrete and conceptual differences between the approaches.

1 Introduction

The current MDA initiative favors the use of model transformations within UML-based development of software systems for a number of different purposes. Model transformations are being applied for establishing consistency of UML models [6], for transforming business models into BPEL [8], for refactoring purposes [17] and for pattern applications.

With model transformations being applied in such diverse scenarios, there is a strong need for techniques and methodologies dealing with developing model transformations. Currently, a lot of research is performed in the direction of how to express model transformations and of building appropriate tool support. The QVT initiative [15] aims at developing a standard for model transformations. However, it is still unclear which model transformation approaches are best suited for which applications and further whether it will be possible to express all possible model transformations with one approach. It seems to be the case that a number of different approaches will be needed in the future. As a consequence, it will be important to know the advantages and disadvantages of the different approaches to help choosing the right approach for a given application.

In this paper, we present and discuss two different model transformation approaches: a relational approach and a graph transformation approach. First, in Section 2, we introduce the running example for our comparison. Then, we focus on a graph transformation approach in Section 3 and on a relational approach in Section 4. We then compare the two approaches in detail in Section 5 and finally draw conclusions for future development of model transformation technology.

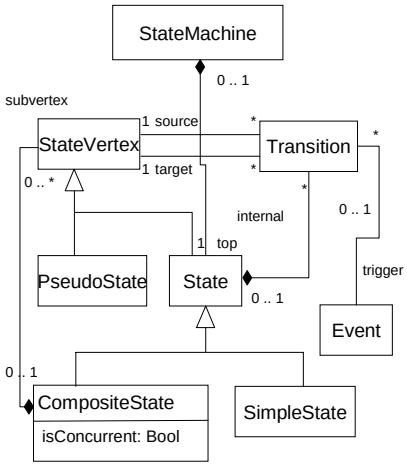

Fig. 1. The metamodel for statecharts

2 Statechart to CSP Translation and Metamodels

One typical approach for defining and checking semantic consistency of a UML model is to translate the relevant parts of the model into a formal language [6]. For checking behavioral consistency, we have developed complex model transformations of UML statecharts to the process algebra called Communicating Sequential Processes (CSP) [10]. After translation of statecharts to CSP, formal consistency conditions can be specified and checked using existing model checking support. Currently, our model transformation approach and tool support (see our research prototype Consistency Workbench [7]) is based on the concept of graph transformation. With the current developments concerning a common standard for model transformations, it will be necessary to know the advantages and disadvantages of different model transformation approaches.

As this paper aims at a comparison of a relational and a graph transformation approach, we will now introduce a common example, a model transformation from UML statecharts to CSP. Since OMG's MOF 2.0 Query/View/Transformation Request for Proposal requires transformation of MOF-compliant models, we introduce the following metamodels, which define the source and target models. In Figure 1, a metamodel for statecharts is shown, taken and adapted from the UML specification.

As a basis for developing the metamodel, we use the following simplified syntax of CSP: Given a set $\mathcal{A}$ of actions (including compound actions, see below) and a set of process names $\mathcal{N}$, the syntax of CSP is given by

$$P ::= \texttt{STOP} \mid \texttt{SKIP} \mid a \rightarrow P \mid \text{if } \mathit{bool} \text{ then } P \text{ else } P \mid pn$$

where $a, b \in \mathcal{A}$, $pn \in \mathcal{N}$, and non-terminal P. Process names are used for defining recursive processes using equations $pn = P$.

The interpretation of the operations is as follows. The processes `STOP` and `SKIP` represent, respectively, deadlock and successful termination. The prefix

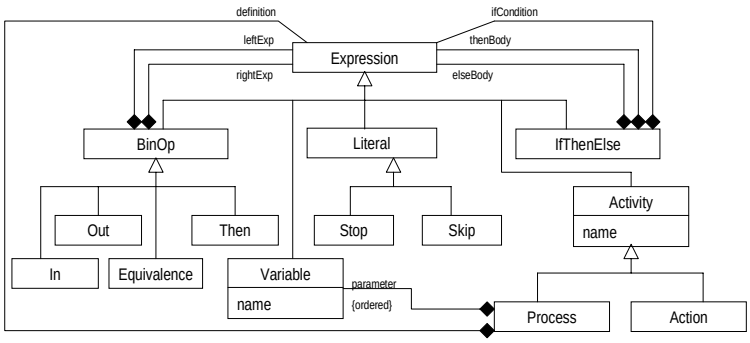

Fig. 2. The metamodel used for CSP

process $a \rightarrow P$ performs action a and continues like P. Note that a can also be a compound action and includes constructs like $c!b$ or $c?b$ for sending or receiving an event b via a channel c, respectively. CSP also includes a so-called conditional choice operator represented as an if-then-else construct.

Note that for simplification we have left out further binary operators such as interleaving or parallel composition. The metamodel shown in Figure 2 is a rather direct translation of the simplified syntax: a CSP expression is a binary operator (together with two expressions) or an if-then-else expression or one of the designated CSP literals STOP or SKIP. Further, an expression can be an activity. Binary operators include the *then* (CSP $\rightarrow$), *out* (CSP !) operator, or *in* (CSP ?) operator.

3 Rule-based Model Transformations with Control Conditions

A transformation of two or more models may be described by specifying how a model conforming to its metamodel is translated into a corresponding model conforming to the other metamodel. The concept of graph transformation has been a traditional candidate for specifying model transformations in an operational way because visual models can be considered as special forms of graphs. The main idea is to specify how elements of one model are transformed into elements of another model, by using a set of transformation rules.

Based on a sound mathematical background, there is a rich theory available for graph transformation (see e.g. [5]). This facilitates the analysis of properties of model transformations such as termination and confluence. From a practical point of view, a number of recent approaches have adapted graph transformation for expressing model transformations on UML models [3] [2]. In the following, we will introduce rule-based model transformations with control conditions (based on [12]) which has been our approach for specifying a complex model transformation from UML to CSP [7].

A model transformation for translating a model from a source language to a target language can be defined by a set of so-called compound rules. Each such compound rule $r : (r_s, r_t)$ consists of two individual rules: The source

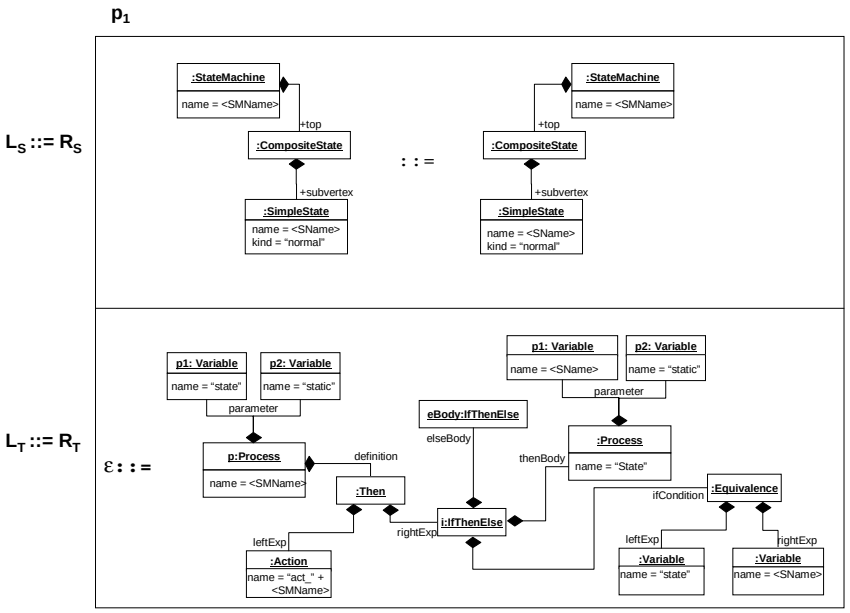

Fig. 3. The first transformation rule for statechart/CSP mapping

transformation rule $r_s : L_S ::= R_S$ describes the transformation of the source model, the target transformation rule $r_t : L_T ::= R_T$ specifies the transformation of the target model.

In Figure 3, a compound rule p_1 for (partially) translating UML statecharts to CSP is shown (based on [12]). The general idea is to create a parameterized CSP process for each state of the state machine. The compound rule p_1 creates such a process (called p:Process in the figure) together with an ifThenElse which branches to the process of the simple state. To achieve this, p_1 consists of two parts, one for matching the state machine (shown in the upper part) and one for creating the corresponding CSP process (shown in the lower part). More formally, in our approach each rule $r : (r_s, r_t)$ consists of a UML part and a CSP part. Concerning p_1 in the figure, r_s is the UML part in the upper part of the figure, r_t is the CSP part in the lower part of the figure. Both the r_s and r_t can be viewed as graph transformation rules when interpreting the visual models as attributed typed graphs [9].

Source and target rules are coupled by the ability of using shared variables. Such variables are denoted by $\langle variable \rangle$. These variables are used for being able to transfer model information from one model (e. g. the state machine) to another model (e. g. the CSP model). For example, SMName is such a variable in the compound rule p_1 for creating a CSP process with the name of the state machine.

For translation of a source to a target model, we briefly describe how a compound rule is applied, assuming that $X = \{x_1, .., x_n\}$ is the set of variables of L_S:

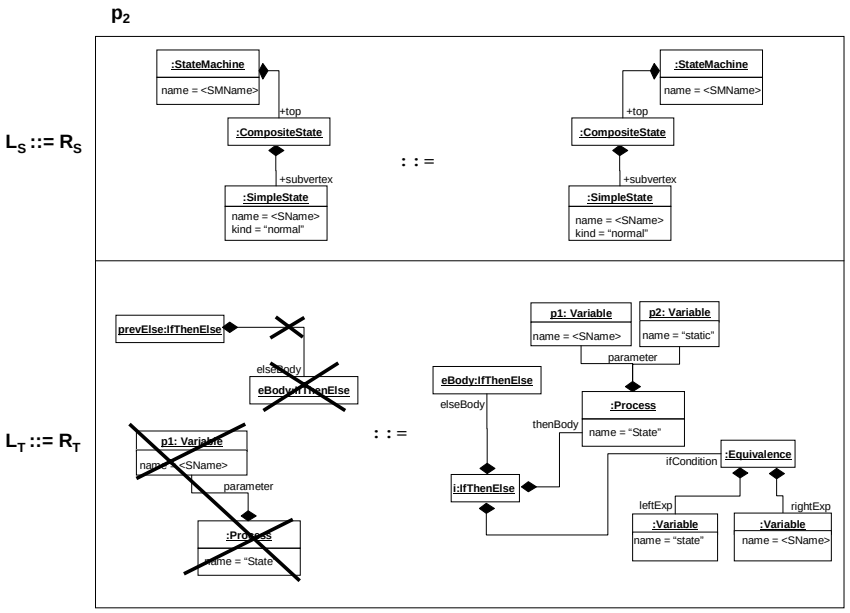

Fig. 4. The second transformation rule for statechart/CSP mapping

1. an occurrence of the left side L_S of the source transformation rule is searched within the source model, such an occurrence is called *source match*.
2. having found a source match, the variables are given concrete values, leading to a variable instantiation denoted X^I.
3. the left side L_T of the target transformation rule is instantiated with the values of the variables, denoted also by $L_T(X^I)$.
4. an occurrence of the instantiated left side of the target transformation rule is searched within the target model, such an occurrence is called *target match*.
5. the right side R_S of the source transformation rule is instantiated with the values of the variables.
6. the right side R_T of the target transformation rule is instantiated with the values of the variables.
7. the occurrence of the instantiated left side of L_S is replaced with the instantiated right side R_S of the source transformation rule
8. the occurrence of the instantiated left side of L_T is replaced with the instantiated right side R_T of the target transformation rule.

Note that we do not go into details here when a compound rule is well-formed. For example, one possible requirement would be that the set of variables used in L_T is contained in the set of variables used in L_S.

In Figure 4 and 5, two further rules are shown, which (together with p_1) specify a partial translation of state machines to CSP as follows: First, p_1 is applied which matches the structure of StateMachine with a top CompositeState and a SimpleState in the model and then creates a corresponding process structure in the CSP model, together with a contained process for the first simple state, and

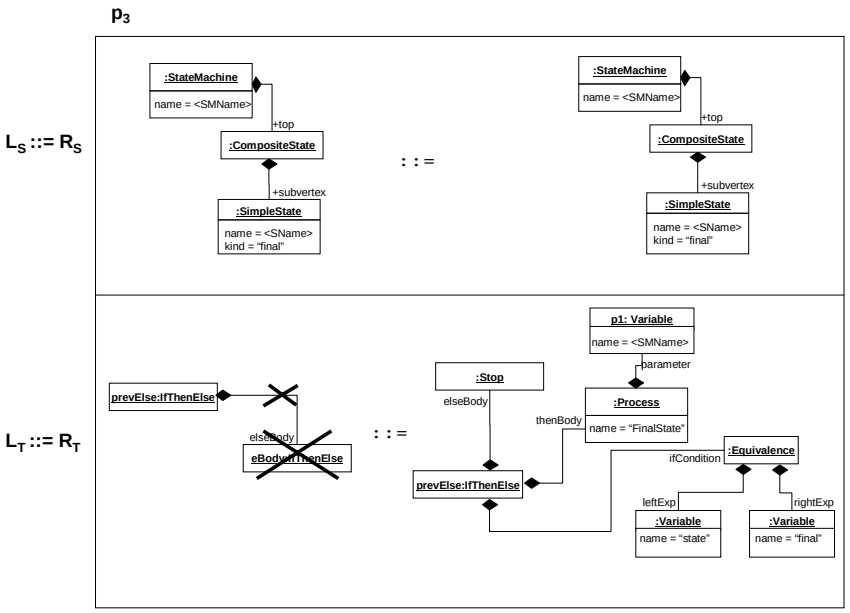

Fig. 5. The third transformation rule for statechart/CSP mapping

an ifThenElse with an empty eBody. Note that here L_T is the empty word, requiring no CSP model to exist. Rule p_2 matches another simple state in the state machine and a corresponding ifThenElse in the CSP model. The negative application condition of L_T (represented by the elseBody link, the eBody:IfThenElse, the :Process and p1:Variable crossed out) ensures that these model elements do not exist and allow to distinguish between simple states already matched in the UML model and those not dealt with. If the rule p_2 is applied, a new thenBody link together with the process for the simple state is created. Additionally, a new placeholder eBody:IfThenElse is created. The last rule p_3 matches one final state and creates a process for this state. In this example, we assume that p_1 is applied only once, then p_2 is applied as long as possible, and then p_3 is applied once.

In order to be able to specify in which order compound rules are applied, they must be assembled to a so-called transformation unit with a control expression. Following existing work on transformation units [11], we will assume that rules are organized in rule sets which are then organized in a sequence of rule sets where each rule set can be considered as a layer. Within a rule set, rules may be applied non-deterministically.

Syntactically, we express layers of rule sets as follows: Assuming the three rules p_1, p_2, p_3, then $\langle p_1, p_2 \downarrow, p_3 \rangle$ specifies three layers, each containing a p_i. This means that first all rules within the first layer are applied and then the ones in the second layer. For each rule, it is indicated whether it is applied once or as long as possible by a simple marker. For example, $p_2 \downarrow$ denotes that the rule p_2 is iterated until it cannot be applied anymore.

Our concept of rule-based model transformation has been validated in the consistency workbench [7], specifying two large transformation units for translating statecharts and collaborations to CSP.

4 A Relational Model Transformation Approach based on QVT-Merge

In relational approaches, a transformation of two or more models may also be described by *relating* their elements [1]. In this approach, a set of relations specify *what* the transformation changes in the model(s) instead of specifying *how* these changes are computed. The relational approach is in this sense similar to declarative languages like Haskell or logic-based languages like Prolog.

According to their mathematical nature, relations do not imply a direction. Thus, they are suitable for multi-directional transformation purposes. In particular, bi-directional transformations between two models are generally considered important for MDA [13]. For instance, round-trip software engineering benefits from being able to freely move between different levels of model abstraction. As the notion of a relation has its origin in mathematics, this provides a means to define a formal semantics for relational approaches.

In bi-directional transformation, care has to be taken if information gets lost in a transformation step (if the transformation is a surjective function) or if information is added (if the transformation is an injective function). It is possible to handle some of these problems using tracing techniques, but it is a difficult problem to address in general.

```
relation R {
    domain { pattern_1 when condition_1 }
    ...
    domain { pattern_n when condition_n }
    when { condition }
 }
```

Fig. 6. The syntax of a relation in the current QVT-Merge RFP submission.

There are two different approaches to find the elements that are part of a relation. The first approach - *querying* - evaluates an expression over a model, returning those elements of the model for which the constraint holds. Such queries on models can be formulated in UML's Object Constraint Language (OCL). The other common approach uses *pattern matching* where a term or a graph pattern containing free variables is matched against the model.

Relations are not executable; to actually execute a relation-based transformation, a number of approaches exist. The QVT-Merge Group [14] proposes a hybrid approach that suggests to add imperative constructs (called *mappings*)

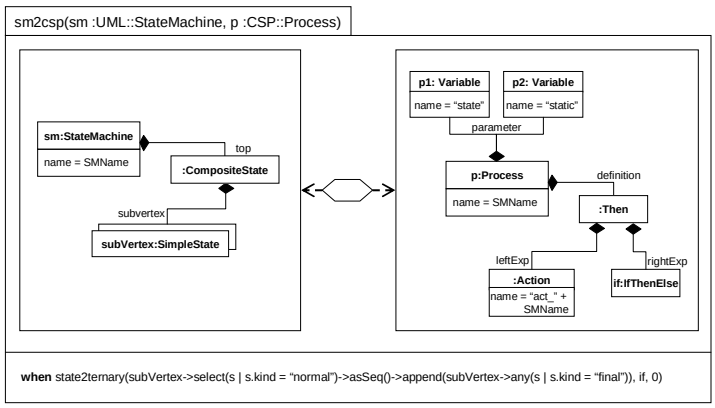

Fig. 7. The top level relation *sm2csp*.

that implement the specification defined by the relations. Nevertheless it is possible to give their relational language an executable semantics by treating it as a constraint satisfication problem, extended with the ability to delete information for states that cannot satisfy the constraint. Operationally, this approach involves creating, modifying, and/or deleting elements so that the relations are made consistent. In this paper, since we are investigating relational approaches, we will take this approach using the relational language of QVT-Merge.

In general, the relational approach starts with one relation between the top level elements of the models, in which child relations on the models contents are established.

The skeleton of an exemplary relation according to the current status of the QVT-Merge Group's proposal is shown in Figure 6. The proposal defines a pattern matching language. A relation has a unique name (R) by which it may be referenced by another relation as a child relation. For each model that is part of the relation, a domain is defined that contains the pattern expression on the respective meta model. The scope of each domain may be constrained by a condition. Domains may be related to each other implicitly, using variables common with other domains, or explicitly, using constraints in a separate condition labelled with the *when* keyword, which scopes over all domains.

In the following, we will illustrate a relational approach of transforming a statechart into the CSP language. The relations will be presented in the diagrammatic form introduced in the QVT-Merge proposal, where the domains are depicted graphically and the respective *when* condition is given in a textual form.

The relation *sm2csp* (Figure 7) maps the skeletons of the state machine on the left hand side and the CSP process on the right hand side. LHS and RHS are mapped by graphical pattern expressions where the elements are bound to each other by corresponding variable names. The variable names used are the name of the statechart *SMName* and the sequence of vertices *subVertex* on the LHS; on the RHS, the top level *IfThenElse* node is referenced by the variable *if* because this reference is needed in the *when* condition. The structure of the

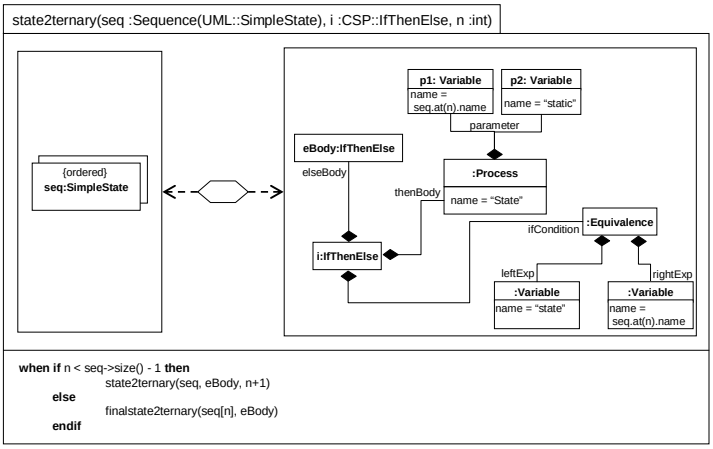

Fig. 8. The relation *state2ternary* contains a recursive invocation.

statechart is mapped to CSP by invoking the sub-relation *state2ternary* on the sequence of subvertices (captured by the OCL select expression) in the *when* condition of the relation.

The recursive mapping of the states to ternary CSP constructs is defined in the relation *state2ternary* (Figure 8). The *when* condition of this relation deals with the control flow of the relation: if the next state in the sequence of states is the last element, the relation *finalstate2ternary* (Figure 9) is invoked. Otherwise, the relation *state2ternary* is invoked recursively. Note that this condition makes use of the assumption that the final state is the last element in the sequence (as formed by the OCL select expression, given in the *when* condition of Figure 7).

The transformation of a statechart's final state is covered by the relation *finalstate2ternary* (Figure 9) that maps the final state of the statechart to the last *IfThenElse* construct in the CSP program (cf. Figure 5).

5 Comparison and Discussion

A comparison of the graph transformation approach and the relational approach can be performed under different criteria.

In general, one can compare the two approaches from a software engineering point of view. One important criterion for model transformation technology to succeed is certainly its usability. Here we understand by usability the ease of designing a model transformation as well as the amount of effort for the transformation specifier to design rules (e.g., in some approaches, the same model transformation may lead to more concise rules than in others). With regard to the concrete comparison, we observe that in both approaches we deal with approximately the same number of rules. In addition, even the models in the rules are rather similar. A slight difficulty may be seen in the concept of compound rules for graph transformation whereas in a relational approach the knowledge of the additional expression language for conditions is required. Usability also

depends on the background of the users and will probably require extensive empirical studies before one can make any major conclusions in this direction.

The two approaches can also be compared from a computational viewpoint. Here, one focuses on how the individual transformations will be computed by a transformation engine. This can lead to efficiency measures as well as an insight about the capability of the transformations. For comparing the two approaches, we will introduce what operations are required by a machine for realizing each approach. First, for the graph transformation approach, the machine must be capable of performing the following operations:

- *match(Model m, Pattern p)* matches the pattern p on the model m and returns a match mt.
- *replace(Model m, Match mt, Replacement r)* replaces the match mt with the replacement r in the model m. Note that replace includes possible deletion of elements.

Second, for the relational approach, the machine must be capable of performing the following operations:

- *matchRooted(Model m, Pattern p, Element r)* matches the pattern p rooted in r on the model m and returns a match mt.
- *reconcile(Model m1, Model m2, Match mt1, Match mt2, Relation r)* reconciles $mt1$ and $mt2$ with respect to relation r in models $m1$ and $m2$. This may involve creating, modifying, and deleting elements so that the constraint is satisfied. In other words, this operation performs constraint solving with possible deletion.

For the above operations, different implementations can be thought of, depending on the underlying representation of the model and the pattern: Either graph matching or string matching, if the pattern and model are represented as string. Note that for the relational approach, the matching algorithm is assisted by rooting it with an element of the model. This decreases the complexity of the matching which has been a theoretical problem in graph matching. Note however, that also for graph matching in general there are efficient techniques [4].

Using these operations, we are capable of expressing the operations of both approaches. For the graph transformation approach, we get the following algorithm which corresponds to our description in Section 3.

```
graph_transformation(Source sourcemodel, target targetmodel, compound rule c)

begin
  sourcematch = match(sourcemodel, c.getSourcePattern())
  targetmatch = match(targetmodel, c.getTargetPattern())
  if sourcematch.result AND targetmatch.result then
   sourcereplacement = c.getSourceReplacement(matchsource)
   targetreplacement = c.getTargetReplacement(matchtarget)
   replace(sourcemodel,sourcematch, sourcereplacement)
```

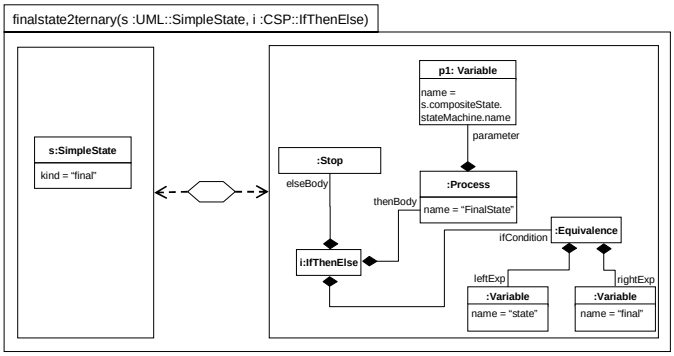

Fig. 9. The final state is dealt with in the relation *finalstate2ternary*.

```
   replace(targetmodel,targetmatch, targetreplacement)
  else
    doNothing()
  endif
end
```

We additionally assume simple helper operations such as getSourcePattern and getTargetPattern.

For the relational approach, we can also express the algorithm as follows:

```
relational(Source sourcemodel, target targetmodel, relation r)

begin
  sourcematch = matchRooted(sourcemodel, r.leftPattern, r.args[0])
  targetmatch = matchRooted(targetmodel, r.rightPattern, r.args[1])
  reconcile(sourcemodel, targetmodel, sourcematch, targetmatch, r)
end
```

In principle, the two algorithms have a similar form and have similar kinds of operations. We will explain the differences in the following. Note that the following will be a rather intuitive approach, leaving a detailed complexity analysis to a more elaborated version of this paper.

The matching procedures are similar in each approach, both requiring a graph to be matched against the corresponding model. The relational approach to matching offers a rooted match, which means that the graph to be matched always has an element in the model upon which it can fix the match. Rooted matching can be more efficiently performed than their non-rooted counterparts. In the worst, general case, non-rooted matches are less efficient by a factor equivalent to the number of nodes in the model. The relational approach clearly depends on the reconciliation procedure. We do not go into the details of this activity, as there are many possible ways to realize it. Nevertheless, the rule specifier must have some knowledge as to how reconciliation is performed for

the approach to be usable, in general. With regard to the graph transformation approach, the replacement can be thought of as a specific way of reconciliation. In practice, it will be important that the rule specifier has a clear idea of what will be the result of its rule specification. This is currently the case for graph transformation but not for the relational approach.

When regarding the rules presented in Section 3 and Section 4, it becomes obvious that in using the relational approach the rule specifier has tackled the problem in a similar way to the method taken with the graph transformation approach:

- Firstly, the problem involved mapping UML statemachines to CSP, and thus in using both approaches, only one direction of transformation was considered. Even in the relational approach, the specifier has thought about which CSP model elements should be created for a given UML model element.
- Secondly, in the relational approaches, the specifier has used a reconciliation approach that favors the matching of one specific model element in the CSP part and the creation of additional model elements in the CSP part: The specifier has coupled rules *sm2csp* and *state2ternary* in the way that the IfThenElse is created by *sm2csp* and then matched in *state2ternary*. Thereafter, the other model elements of the CSP part will be created.

Currently, it is unclear whether it is possible for a rule specifier to use the power of reconciliation in a more general form.

Another aspect is the problem of bi-directionality: In contrast to the graph transformation approach, a relational approach enables the software engineer to think also bidirectionally, although this is sometimes more difficult and involves more work. Although bidirectionality is possible as well in graph transformation approaches (see the Triple Graph Grammar approach by Schuerr [16]), the general rule format does not force the rule specifier to think about it (one might as well simply specify uni-directional transformations). For example, the rules in Section 3 cannot be used for CSP to UML statechart translation without modification.

6 Conclusion

The success of model transformation technology will depend on a common language for designing them or, alternatively, on the existence of many languages for specific model transformations used in various domains. In the second case, detailed characteristics are needed which help to decide which approach to use in a certain application. With the current amount of model transformation approaches available, the question arises which is best suited for which application and where the differences are. In this paper, as a first step towards a general classification of approaches, we have compared two model transformation approaches, a graph transformation approach and a relational approach.

After introducing a running example for the comparison, we have expressed transformation rules in both approaches. The comparison has then shown that

there are some similarities between both approaches. Graph transformation approaches favor matching and replacement, the relational approach favors matching and reconciliation. By providing algorithms in pseudo-code, we have been able to detail the differences: Firstly, the matching is more general in the graph transformation approach. Secondly, replacement in the graph transformation can be thought of a specific way of reconciliation.

From our comparison we can draw the following conclusions: In principle, the two approaches are rather similar. The graph transformation approach has the power in the clear operational idea which enhances rule specification. The relational benefits from the bidirectionality idea. As a special form of reconciliation, namely replacement of model elements, the graph transformation approach can be seen as one implementation of a relational approach. Further, it remains to be shown in practice that a human mind is capable of using different ways of reconciliation when designing the rules. Another important conclusion from the two approaches being rather similar is that results from graph transformation theory can also be applicable to a relational approach. These may include insights about confluence of model transformations or efficient ways of implementing the graph matching part.

References

1. D. Akehurst and S. Kent. A Relational Approach to Defining Transformations in a Metamodel. In Jean-Marc Jezequel and Heinrich Hussmann, editors, *UML 2002 - The Unified Modeling Language: Model Engineering, Concepts, and Tools*, volume 2460 of *Lecture notes in computer science*. Springer, October 2002.
2. P. Braun and F. Marschall. BOTL - The Bidirectional Object Oriented Transformation Language. Technical report, Fakultät für Informatik, Technische Universität München, Technical Report TUM-I0307, 2003.
3. G. Csertán, G. Huszerl, I. Majzik, Z. Pap, A. Pataricza, and D. Varró. VIATRA: Visual Automated Transformations for Formal Verification and Validation of UML Models . In *Proceedings 17th IEEE International Conference on Automated Software Engineering (ASE 2002)*, pages 267–270, Edinburgh, UK, September 2002.
4. H. Dörr. *Efficient Graph Rewriting and Its Implementation*. LNCS 922, Springer-Verlag, Berlin, Germany, 1995.
5. H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*. World Scientific, 1999.
6. G. Engels, R. Heckel, and J. M. Küster. Rule-Based Specification of Behavioral Consistency Based on the UML Meta-model. In M. Gogolla and C. Kobryn, editors, *UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools., 4th International Conference, Toronto, Canada, October 1-5, 2001, Proceedings*, volume 2185 of *LNCS*, pages 272–287. Springer-Verlag, 2001.
7. G. Engels, R. Heckel, and J. M. Küster. The Consistency Workbench - A Tool for Consistency Management in UML-based Development. In P. Stevens, J. Whittle, and G. Booch, editors, *UML 2003 - The Unified Modeling Language. Modeling Languages and Applications. 6th International Conference, San Francisco, October 20 -24, USA, Proceedings*, volume 2863 of *LNCS*, pages 356–359. Springer-Verlag, 2003.

8. R. Hauser and J Koehler. Compiling Process Graphs into Executable Code. In *Proceedings of the 3rd International Conference on Generative Programming and Component Engineering,*. Springer-Verlag, October 2004. To appear.
9. R. Heckel, J. M. Küster, and G. Taentzer. Towards Automatic Translation of UML Models into Semantic Domains. In H.-J. Kreowski and P. Knirsch, editors, *Proceedings of the Appligraph Workshop on Applied Graph Transformation*, pages 11–22, March 2002.
10. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
11. S. Kuske. *Transformation Units - A Structuring Principle for Graph Transformation Systems*. Dissertation, Universität Bremen, 2000.
12. J. M. Küster. *Consistency Management of Object-Oriented Behavioral Models*. PhD thesis, University of Paderborn, March 2004.
13. Object Management Group. Model driven architecture, 2001. http://www.omg.org/mda.
14. Object Management Group (OMG). *QVT-Merge Group. MOF 2.0 Query/Views/Transformations, Revised Submission. OMG document ad/2004-04-01*, April 2004.
15. MOF 2.0 query / views / transformations RFP. OMG document ad/02-04-10, 2002.
16. A. Schürr. Specification of Graph Translators with Triple Graph Grammars. In Tinhofer, editor, *Proceedings WG'94 International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 151–163. LNCS 903, Springer-Verlag, 1994.
17. P. v. Gorp, H. Stenten, T. Mens, and S. Demeyer. Towards Automating Source-Consistent UML Refactorings. In P. Stevens, J. Whittle, and G. Booch, editors, *UML 2003 - The Unified Modeling Language. Modeling Languages and Applications. 6th International Conference, San Francisco, October 20 -24, USA, Proceedings*, volume 2863 of *LNCS*, pages 144–158. Springer-Verlag, 2003.