Generating Transformation Rules from Examples for Behavioral Models

Marouane Kessentini[1], Manuel Wimmer[3], Houari Sahraoui[1] and Mounir Boukadoum[2]
[1]DIRO, Université de Montréal,Canada
{kessentm,sahraouh}@iro.umontreal.ca
[2]DI, Université du Québec a Montréal, Canada
Mounir.boukadoum@uqam.ca
[3]Vienna University of Technology, Austria
wimmer@big.tuwien.ac.at

ABSTRACT
Behavioral UML models like sequence diagrams (SD) lack sufficient formal semantics, making it difficult to build automated tools for their analysis, simulation and validation. A common approach to circumvent the problem is to map these models to more formal representations. In this context, many works propose a rule-based approach to automatically translate behavioral models like SD into colored Petri nets (CPN). However, finding the rules for such SD-to-CPN transformations manually may be difficult, as the transformation rules are usually not obviously defined. We propose a solution that starts from the hypothesis that examples of good transformation of SD-to-CPN can be useful to automatically generate transformation rules. To this end, we describe an automated approach to find the rules that best match the meta-model elements of SD to corresponding elements in the CPN meta-model. Thus, our approach starts by randomly generating a set of rules, executing them to generate some target models. Then, it evaluates the quality of the proposed solution (rules) by comparing the generated target models to the expected ones in the base of examples. In this case, the search space is large and heuristic-search is needed. To achieve our goal, we combine two algorithms for global and local search, namely Particle Swarm Optimization (PSO) and Simulated Annealing (SA). Our empirical results show that the generated rules derive CPNs similar to the expected ones.

Categories and Subject Descriptors
D.2.10 [**Design**]: Methodologies

General Terms
Design

Keywords
Rules generation, Model transformation, Search-based software engineering, Behavioral models

1. INTRODUCTION
Model Transformation plays an important role in Model Driven Engineering (MDE) [1]. The research efforts by the MDE community have produced various languages and tools, such as ATL [2], KERMETA [3] and VIATRA [4], for automating transformations between different formalisms. One major challenge is to automate these transformations while preserving the quality of the produced models [1, 6].

BM-FA '10, June 15, 2010, Paris, France

Many transformation contributions target UML models [1, 6]. From a transformation perspective, UML models can be divided into two major categories: structural models, such as class diagrams, and behavioral models, such as activity, sequence diagrams (SD) and state diagrams [7]. Models of the second category are generally transformed for validation and simulation purposes. This is because UML behavioural models lack sufficient formal semantics [8]. This limitation makes it difficult to build automated tools for the analysis, simulation, and validation of those models [9]. A widely accepted approach to circumvent the problem uses concomitant formal representations to specify the relevant behavior [11]; Petri Nets (PNs) [10] are well suited for the task. PNs can model, among others, the behavior of discrete and concurrent systems. Unlike SDs, PNs can derive new information about the structure and behavior of a system via analysis. They can be validated, verified, and simulated [11]. Moreover, they are suitable for visualization (graphical formalism) [11]. These reasons motivate the work to transform UML SDs to PNs.

SD-to-PN transformation may be not obvious to realize, due to different reasons [29]. In fact, defining transformation rules can be difficult since the source and target languages have elements with different semantics; therefore, 1-to-1 mappings are not sufficient to express the semantic equivalence between meta-model elements. Indeed, in addition to ensuring structural (static) coherence, the SD-to-PN transformation should guarantee behavioral coherence in terms of time constraints and weak sequencing. For instance, the transformation of a SD message depends on the order (sequence) inside the diagram and the events within different operands (parallel merge between the behaviors of the operands, choice between possible behaviors, etc.). One solution of these problems, related to rules definition, is to propose a semi-automated approach for rules generation in order to help the designer.

In this paper, we explore a solution for automating rules generation, based on the hypothesis that valid transformations of SD-to-PN, called transformation examples, can be used to generate transformation rules. In this context, our approach, inspired by the Model-Transformation-by-Examples (MTBE) school [12, 13, 14, 15, 30], helps rules generation. The motivation for transformation-by-examples approaches is that the experts may find it difficult to master both the source and target meta-models. On the other hand, it is recognized that experts can more easily give transformation examples than complete and consistent transformations rules [2]. These transformation examples are

exploited to generate (1) rules using inductive logic programming (ILP) [13], or (2) directly the target model without rules generation, using optimization techniques [14, 15]. For both approaches, an example corresponds to interrelated source and target models (traces). However, generating traces is a fastidious task because the expert needs to manually define them. In addition, it is difficult for an expert to explain his reasoning process, in form of traces, when transforming a source model.

We propose to alternatively view transformation rules generation as an optimization problem where rules are automatically derived from available examples. In our case, each example corresponds to a source model and its corresponding target model without transformation traces. Then, our contribution, called SERGE (SEarch-based Rules Generation by Example), starts by randomly generating a set of rules, executing them to generate some target models, and then evaluates the quality of the proposed solution (rules). It compares the generated target models and expected ones in the base of examples. Due to the large number of possible matchings between source and target meta-models elements, a heuristic-search strategy is used to build the solution. To achieve our goal, we combine two algorithms for global and local search, namely Particle Swarm Optimization (PSO) and Simulated Annealing (SA). Our empirical results show that the generated rules derive CPNs similar to the expected ones.

The remainder of this paper is structured as follows. In section 2, we provide an overview of the proposed approach for generating SD-to-CPN transformations rules and discuss its rationale in terms of problem complexity. Section 3 describes the heuristic algorithm based on the combined PSO and SA search heuristics. An evaluation of the algorithm is explained and its results are discussed in Section 4. Section 5 is dedicated to the related work. Finally, concluding remarks and future work in section 6.

2. APPROACH OVERVIEW

A model transformation takes a model to transform as input, the *source model*, and produces another model as output, the *target model*. In our case, the source model is a UML sequence diagram and the target model is a colored Petri net. We describe, in this section, the principles of our approach and discuss the rationale behind given the complexity of the transformation problem.

2.1 Overview

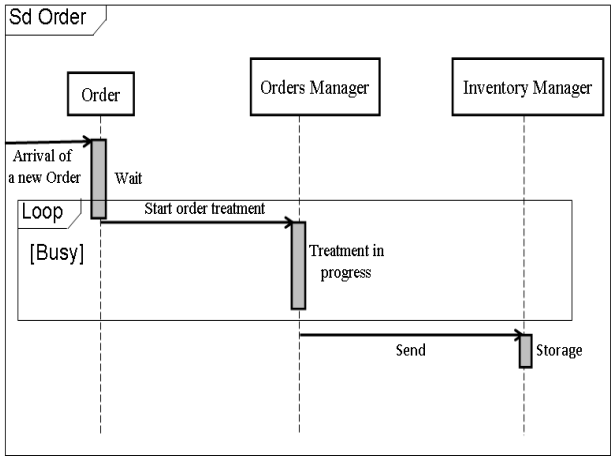
SERGE takes as input two types of information: the source and target metamodels (SD and CPN in our case), and a set of transformation examples. Each example is defined as a pair of models. Then, it generates a set of transformation rules as output. The source and target metamodels allows to randomly generate the rule sets. The quality of the produced rule sets is measured by the level of their conformance with the example base. Indeed, each candidate rule set is applied to the source models in the base and the produced target models are compared to the expected target models.

As many rule sets are possible, the generation process is a combinatorial optimization problem where the dimensions of the search space are the elements of the SD metamodel. A solution is determined by the assignment of one or many elements in the CPN meta-model to each SD element. The search is guided by the quality of the solution described bellow.

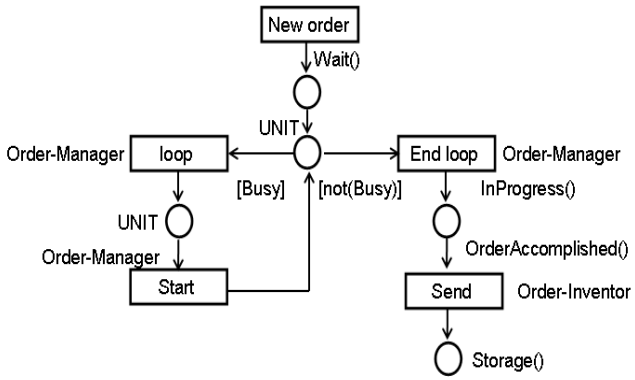
To explore the solution space, the search is performed in two steps. First, we use a global heuristic search by means of the PSO algorithm [18] to reduce the search space size and select a first transformation rule set. Then, a local heuristic search is done using the SA algorithm [19] to refine this solution. In order to provide the details of our approach, we define some terms.

A ***transformation example*** (TE) is SD and his equivalent CPN without transformation links (traces). Formally, we view a TE as a triple <*SMD, MB*> where SMD denotes the source model (SD in our case), TMD denotes the target model (CPN in our case).

The ***base of examples*** is a set of transformation examples. The transformation examples can be collected from different experts.



(a)



(b)

Figure 1. Transformation example.

A TE of a SD-to-CPN is presented in Figure 1. The models (source or target) are described by sets of predicates, each corresponding to a construct (or a sub-construct) type. The predicate types for SDs are:

```
Object (ObjetName, ObjetType);
Message (MessageType, Sender, Receiver,
  MessageName, ActivityName);
Activity (ActivityName, ObjectName, Duration,
  MessageNumber);
Loop (StartMessageName, EndMessageName,
  ConditionValue);
Par (StartMessageName, EndMessageName,
  ConditionValue, ConditionType);
```

Similarly, those of CPN are:

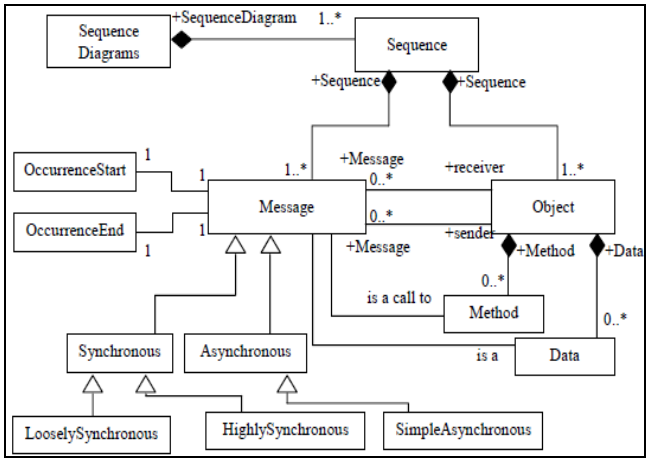
```
Place (PlaceName);
Transition (TransitionName);
Arc(Input(TransitionName,PlaceSourceName),
Output(TransitionName, PlaceDestinationName));

Action(Name, Type);
```

For example, the message *Arrival of a new Order* in Figure 1.a can be described by

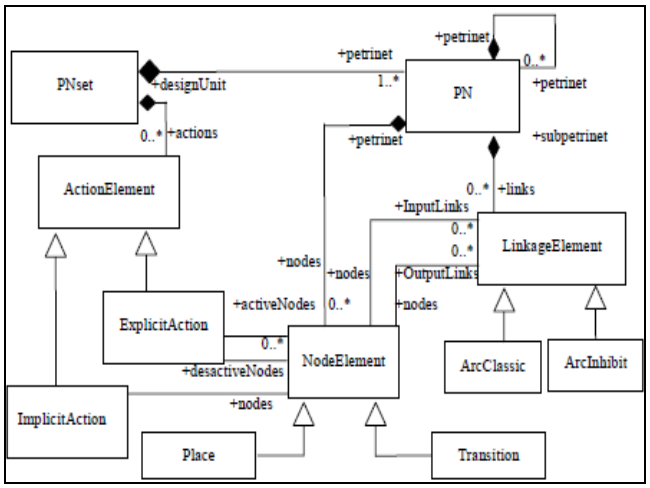
```
Message (Synchronic,_, Order, ArrivalOfNewOrder,
  Wait);
```

The predicate indicates that *Arrival of a new Order* is a synchronic message sent to Order (with "_" meaning no sender) and connected to activation box *Wait*.

The solution proposed by the heuristic search algorithm is a matching between the elements in SD and CPN meta-models. Figure 2 shows a simplified version of the two meta-models. The elements of the SD meta-model are: Object; Synchronous Message; Asynchronous Message; Activity; Loop; Par. Similarly, those of CPN are: Place; Transition; Arc Classic; Arc Inhibit; Action;



(a)



(b)

Figure 2. SD and PN meta-models [29].

2.2 Problem Complexity

Our approach assigns to each element in the source meta-model a corresponding element(s) in the target meta-model. Thus, the rules generation process consists of finding the best matching combination between n elements of the SD meta-model and m elements of the CPN meta-model. In this context, $(m!)^n$ possible combinations have to be explored. In fact, we can have one-to-many matching possibilities and not only one-to-one. This value can quickly become huge. A SD meta-model with 6 elements and a CPN meta-model with 5 elements necessitates exploring at least 120^6 combinations. Considering these magnitudes, an exhaustive search cannot be used within a reasonable time frame. This motivates the use of a heuristic search if a more formal approach is not available or hard to deploy.

3. SEARCH-BASED RULES GENERATION BY EXAMPLE

We describe in this section the adaptation of two heuristics, PSO [18] and SA [19], to generate SD-to-CPN transformation rules. These methods each follow a generic strategy to explore the search space. When applied to a given problem, they must be specialized by defining: (1) the coding of solutions, (2) the operators that allow moving in the search space, and (3) the fitness function that measures the quality of a solution. In the remainder of this section we start by giving the principles of PSO and SA. Then, we describe the three above-mentioned heuristic components.

3.1 PSO and SA Algorithms

To obtain a more robust optimization technique, it is common to combine different search strategies in an attempt to compensate for deficiencies of the individual algorithms [20]. In our context the search for a solution is done in two steps. First, a global search with PSO is quickly performed to locate the portion of the search space where good solutions are likely to be found. In the second step, the obtained solution is refined with a local search performed by SA.

PSO, *Particle Swarm Optimization,* is a parallel population-based computation technique proposed by Kennedy and Eberhart [18]. The PSO swarm (population) is represented by a set of K particles (possible solutions to the problem). A particle i is defined by a position coordinate vector X_i, in the solution space. Particles improve themselves by changing positions according to a velocity function that produces a translation vector. The improvement is assessed by a fitness function.

The particle with the highest fitness is memorized as the global best solution (*gbest*) during the search process. Also, each particle stores its own best position (*pbest*) among all the positions reached during the search process. At each iteration, all particles are moved according to their velocities (Equation 1). The velocity V_i' of a particle i, depends on three factors: its inertia corresponding to the previous velocity, its *pbest*, and the *gbest*. Factors are weighted respectively by W, C_1, and C_2. The importance of the local and global position factors varies and is set at each iteration by a random function. The weight of inertia decreases during the search process. The derivation of V_i' is given by Equation 2. After each iteration, the individual *pbest*s and the *gbest* are updated if the new positions bring higher qualities than the ones before.

$$X_i' = X_i + V_i' \quad (1)$$

$$V_i' = W \times V_i + C_1 \times rand() \times (pbest_i - X_i) + C_2 \times rand() * (gbest - X_i) \quad (2)$$

The algorithm iterates until the particles converge towards a unique position that determines the solution to the problem.

Simulated Annealing (SA) [19] is a local search algorithm that gradually transforms a solution following the annealing principle used in metallurgy. Starting from an initial solution, SA uses a pseudo-cooling process where a pseudo temperature is gradually decreased. For each temperature, the following three steps are repeated for a fixed number of iterations: (1) determine a new neighboring solution; (2) evaluate the fitness of the new solution; (3) decide on whether to accept the new solution in place of the current one based on the fitness function and the temperature value. Solutions are accepted if they improve quality. When the quality is degraded, they can still be accepted, but with a certain probability. The probability is high when the temperature is high and the quality degradation is low. As a consequence, quality-degrading solutions are easily accepted in the beginning of process when the temperatures are high, but with more difficulty as the temperature decreases. This mechanism prevents reaching a local optimum.

3.2 PSO and SA for Rules Generation

To adapt PSO and SA to the SD-to-CPN transformation problem, we must define the following: a solution coding suitable for the transformation rules generation problem, a neighborhood function to derive new solutions, and a fitness function to evaluate these solutions.

As stated in Section 2, we model the search space as an *n*-dimensional space where each dimension corresponds to one of the *n* elements of the SD meta-model. The algorithm starts by generating a list of combination sets C_i of CPN meta-model elements. A solution is then a point in that space; defined by a coordinate vector whose elements are combination numbers assigned to the *n* SD-meta-model elements. For instance, the SD meat-model shown in Figure 2 will generate a 6-dimensional space. One solution is this space, shown in Table 1, suggests that an *object* should be transformed according to C_{11} that corresponds, for example, to a *transition* and *place* combination. Thus concretely, a solution (set of rules) is implemented as a vector where the SD meta-model elements are the dimensions and combination numbers are the element values. Thus, each generated rules correspond to: If (SD meta-models element(s)) then (combination of CPN meta-models element(s)).

The association between a SD meta-model element and a CPN meta-model element(s) does not necessarily mean that the match is possible. This is determined by the fitness function described later.

The proposed coding is valid for both heuristics. In the case of PSO, as an initial population, we create *k* solution vectors with a random assignment of the possible combination. Alternatively, SA starts from the solution vector produced by PSO.

Table 1. Solution representation

Dimensions	Meta-model elements	Combination numbers
1	Object	C19
2	Synchronous Message	C17
3	Asynchronous Message;	C51
4	Activity;	C105
5	Loop	C16
6	Par	C83

Change Operators. Modifying solutions to produce new ones is the second important aspect of heuristic search. Unlike coding, change is implemented differently by the PSO and SA heuristics. While PSO sees change as movement in the search space driven by a velocity function, SA sees it as random coordinate modifications.

In the case of PSO, a translation (velocity) vector is derived according to equation 2 and added to the position vector. For example, the solution of Table 1 may produce the new solution shown in Figure 3. The velocity vector *V* has a translation value for each element (real values). When summed with the combination numbers, the results are rounded to integers. They are also bounded by 1 and the maximum number of available combinations.

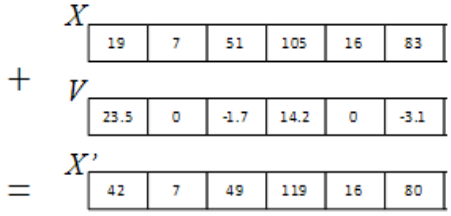


Figure 3. Change Operator in PSO

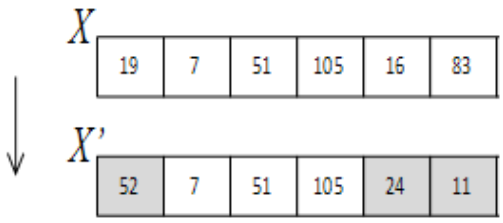


Figure 4. Change Operator in SA

For SA, the change operator randomly chooses *l* dimensions ($l < n$) and replaces their assigned combination numbers by randomly selected ones. For instance, Figure 4 shows a new solution derived from the one of Table 1. Elements 1, 5 and 6 are selected to be changed. They are assigned respectively combination numbers 52, 24, and 11 instead of 19, 16, and 83. The other elements remain unchanged. The number of dimensions to change is a parameter of SA (three in this example).

Fitness Function. The fitness function allows quantifying the quality of a transformation solution. As explained in the previous

paragraph, solutions are derived by random assignment of new combination numbers to the dimensions. Then, the generated solution (rules) is used to transform the source models in the base of examples. The quality of a solution corresponds to the similarity score between the generated target model and the expected ones in the base. Since the models are described in terms of predicates, comparing two CPNs is equivalent to comparing a set of predicates. Thus, the fitness function to maximize is formally defined as follows:

$$f = \sum_{i=1}^{a} \frac{\sum_{k=1}^{p} M_k}{p} \quad (3)$$

Where,

$$M_k = Max_{j=1}^{d}\left(\frac{match_{parameters}(ps, pc_j)}{|parameters(ps)|}\right) \quad (4)$$

$$match_{parameters}(ps, pc_j) = \sum_{o=1}^{m} b_o \quad (5)$$

$$b_f = \begin{cases} 1, \text{ if the } o^{th} \text{ parameter in ps is equivalent to the } o^{th} \text{ parameter in } pc_j \\ 0, \text{ else} \end{cases}$$

a represents the number of examples in the base. *p* is the number of predicates of the CPN for each example. M_k determines the most similar predicates to *ps* by comparison with all *p* predicates in terms of the element types that constitute them. For example, if ps corresponds to Transition (NewPrint, ***Coulor7***), Input(NewPrint, _), Output(NewPrint, *Progress*) and pc_j corresponds to Transition (NewPrint, ***Coulor1***), Input(NewPrint, ***Progress***), Output(NewPrint, Progress), then the similarity score $match_{parameters}$ (ps, pc_j) is equal to 4/6 = 0.66.

The fitness function does not need a considerable effort to be adapted for other transformations (e.g. state machine to PNs).

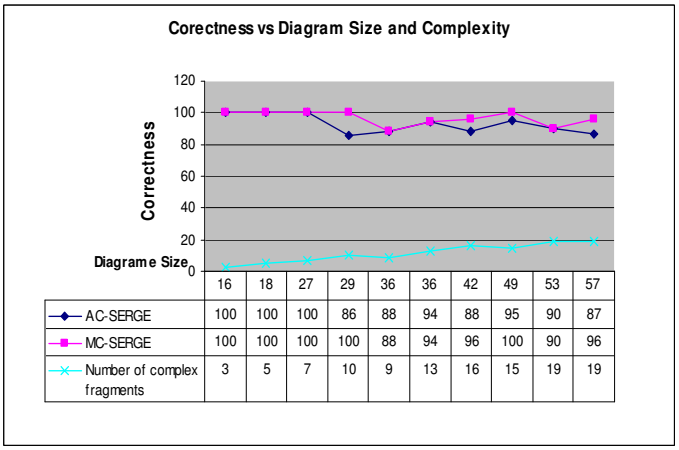
4. EVALUATION

To evaluate the feasibility of our approach, we conducted an experiment on the transformation of 10 UML sequence diagrams. We collected the transformations of these 10 sequence diagrams from the Internet and textbooks and used them to build an example base EB = {<SD_i, CPN_i> | i=1,2,...,10}. We ensured by manual inspection that all the transformations are valid. The size of the SDs varied from 16 to 57 constructs, with an average of 36. The 10 sequence diagrams contained many complex fragments: *loop*, *alt*, *opt*, *par*, *region*, *neg* and *ref*.

To evaluate the correctness of our transformation method, we used a 10-fold cross validation procedure. For each fold, one sequence diagram SD_j is transformed using the remaining 9 transformation examples. Then, the transformation result for each fold is checked for correctness using two methods: automatic correctness (*AC*) and manual correctness (*MC*). Automatic correctness consists of comparing the derived CPN to the known CPN, element by element. This measure has the advantage of being automatic and objective. However, since a given SD_j may have different transformation possibilities, *AC* could reject a valid element transformation because it yields a different CPN from the one provided. To prevent this situation, we also perform manual evaluation of the obtained CPN. In both cases, the correctness is the proportion of elements that are correctly transformed.

Figure 5 shows the correctness for the 10 folds. Both automatic and manual correctness had values greater than 90% in average (92% for AC and 96% for MC). Although few examples were used (9 for each transformation), all the SDs had a transformation correctness greater than 90%, with 4 of them perfectly transformed.

Figure 5 also shows that, in general, the best transformations are obtained with smaller SDs. After 36 constructs, the quality degrades slightly but steadily. This may indicate that the transformation correctness of complex SDs necessitates more examples in general to obtain good rules. However, the largest and most complex SD (57 constructs and 19 complex fragments) has a MC value of 96%.



	16	18	27	29	36	36	42	49	53	57
AC-SERGE	100	100	100	86	88	94	88	95	90	87
MC-SERGE	100	100	100	100	88	94	96	100	90	96
Number of complex fragments	3	5	7	10	9	13	16	15	19	19

Figure 5. Correctness of the generated CPNs.

Manual inspection of the generated CPNs showed that the different transformation errors are easy to fix. They do not require considerable effort and the majority of them is related to the composition of complex fragments.

In conclusion, the quality of the generate CPNs reflect the good quality of the generate rules. In fact, our proposal is to help the designer to define rules but not to fully-automated rules generation. However, the good quality of the generated rules confirms that the designer intervention is very limited and a low effort is needed.

As for execution time, we ran our algorithm on a standard desktop computer (Pentium CPU running at 2 GHz with 2 GB of RAM). The execution time was of the order of a few seconds and increased linearly with the number of examples, indicating that our approach is scalable from the performance standpoint.

5. RELATED WORK

In [5], five categories of transformation approaches were identified: graph-based [22], relational [23], structure-driven [24], direct-manipulation, and hybrid. One conclusion to be drawn from studying the existing MT approaches is that they are often based on empirically obtained rules and it is sometimes difficult to define and express these rules manually [25]. Our approach

SERGE does not usually derive complete and coherence rules, but it is a semi- automated approach to help the designer to define it.

More specifically, in the case of SD-to-PN, several approaches were proposed. In [29], the authors describe a meta-model for the SD-to-PN mapping. It defines rules involving concepts of the meta-models representing respectively sequence diagrams and Petri nets. One of the limitations of this approach is that temporal coherence is not addressed explicitly. Additionally, the meta-model representing the rules tends to generate large PNs, as noticed by the authors. In [11], a set of rules to transform UML 2.0 SDs into PNs is proposed. The goal is to animate SDs using the operational semantics of PNs. Other UML dynamic diagrams are also considered for the transformation to PNs. For example, use case constructs are mapped to PN using a multi-layer technique [8]. The feasibility of this approach was not demonstrated through an implementation. There are other research contributions that concentrate on supporting validation and analysis of UML statecharts by mapping them to Petri nets of various types [36, 37]. A general conclusion on the transformation of behavioural UML models to PNs is that no consensual transformation rules are used.

SERGE uses the "by example" principle to transform models, but what we propose is different from other contributions to model transformation by example (MTBE). Varro and Balogh [12, 13] propose a semi-automated process for MTBE using Inductive Logic Programming (ILP). The principle of their approach is to derive transformation rules semi-automatically from an initial prototypical set of interrelated source and target models. Wimmer et al. [30] derive ATL transformation rules from examples of business process models. Both works use semantic correspondences between models to derive rules, and only static models are considered. Moreover, these works requires traces information to derive rules. These traces are difficult to obtain manually. We have proposed another by example approach, called MOTOE, for model transformation using search-based techniques [14, 15]. MOTOE differs from the two above-mentioned techniques by the fact that it does not derive rules. Rather, it uses examples to transform a specific model. When the model to transform is well covered by the examples, a correct and complete transformation is produced. Otherwise, a partial transformation is proposed anyway.

However, MOTOE is also based on trace information to generate the target model without generating rules. Recently, a similar approach to MTBE, called Model Transformation By Demonstration (MTBD), is proposed [34]. Instead of the MTBE idea of inferring the rules from a prototypical set of mappings, users are asked to demonstrate how the model transformation should be done by directly editing (e.g., add, delete, connect, update) the model instance to simulate the model transformation process step by step. This approach needs a large number of simulated patterns to give good results and, for instance, MTBD cannot be useful to transform an entire source model.

Our approach is inspired by contributions in Search-Based Software Engineering (SBSE) [31, 33]. As the name indicates, SBSE uses a search-based approach to solve optimization problems in software engineering. Once a software engineering task is framed as a search problem, by defining it in terms of solution representation, fitness function and solution change operators, there are many search algorithms that can be applied to solve that problem. To the best of our knowledge, inspired among others by the road map paper of Harman [33], the idea of treating model transformation as a combinatorial optimization problem to be solved by a search-based approach was not studied before our proposal in [14]. For this reason, we can not compare our approach to existing works in SBSE because the application domain is very different.

6. CONCLUSION

In this paper, we propose the SERGE approach to automate SD-to-CPN transformation rules generation using heuristic search. SERGE uses a set of existing transformation examples and the source and target meta-models to derive a set of rules. It starts by randomly generating a set of rules, executing them to generate some target models, and then evaluates the quality of the proposed solution (rules) by comparing the generated target models and expected ones in the base of examples. Our approach differs from rule-based transformation approaches that transform SDs into CPNs as it does not require writing rules. It also differs from exiting by example approaches by the fact that no traceability links are needed in the examples.

We have evaluated our approach on ten sequence diagrams. The experimental results indicate that the derived CPNs are comparable to those defined by experts in the base of examples in terms of correctness (average value of 96%).

A validation on a larger example base is in project to better assess the adaptation capability of the approach. Furthermore, we need to close the loop with the simulation and analysis of the derived CPNs. In a broader perspective, we plan to experiment and extend SERGE to other transformations involving behavioral models: code generation (model-to-code), refactoring (code-to-code), or reverse-engineering (code-to-model), model evolution (model-to-model).

7. REFERENCES

[1] France, R., Rumpe, B.: Model-driven Development of Complex Software: A Research Roadmap. ICSE 2007 : Future of Software Engineering. (2007)

[2] Jouault, F., Kurter, I.: Transforming models with ATL. In: Proc. Of the Model Transformations in Practice Workshop at MoDELS 2005, Jamaica (2005)

[3] Muller, P., F.Fleurey, et J. M. Jezequel (2005). Weaving Executability into Object-Oriented Meta-languages. Proc. of MoDELS' 05, Montego Bay, Jamaica, pp 264-278.

[4] Varro, D., Pataricza, A.: Generic and meta-transformations for model transformation engineering. UML 2004. LNCS, vol. 3273. Springer, Heidelberg (2004)

[5] Czarnecki, K., Helsen, S.: Classification of model transformation approaches. In: OOSPLA 2003, Anaheim, USA (2003)

[6] Ehrig, Hartmut; Ehrig, Karsten; de Lara, Juan; Taentzer, Gabriele; Varró, Dániel; Varró-Gyapay, Szilvia: Termination Criteria for Model Transformation. *Vol. 3442FASE 2005*

[7] Booch, Grady, Jacobson, Ivar and Rumbaugh,James: "The Unified Modeling Language Users Guide", Addison Wesley 1998.

[8] J. Saldhana and S. M. Shatz. UML Diagrams to Object Petri Net Models: An Approach for Modeling and Analysis. *SEKE*, pages 103–110, July 2000.

[9] J. Lilius and I. P. Paltor. vUML: A Tool for Verifying UML Models. ASE99.

[10] T. Murata. Petri Nets: Properties, Analysis, and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[11] Ribeiro OR, Fernandes JM; Some Rules to Transform Sequence Diagrams into Coloured Petri Nets, CPN2006, Jensen K (ed.), Aarhus, Denmark, pp. 237-56, Oct/2006.

[12] D. Varro. Model transformation by example. In Proc.MODELS 2006, pp. 410–424.

[13] D. Varro and Z. Balogh, Automating Model Transformation by Example Using Inductive Logic Programming, ACM Symposium, 2007 (SAC 2007).

[14] M. Kessentini, H.Sahraoui and M.Boukadoum Model Transformation as an Optimization Problem. In Proc.MODELS 2008, pp. 159-173 Vol. 5301 of LNCS. Springer, 2008.

[15] M. Kessentini, H.Sahraoui and M.Boukadoum, Search-based Model Transformation by Example. Submitted to SoSym (under review)

[16] Salvatore Distefano, Marco Scarpa, Antonio Puliafito: Software Performance Analysis in UML Models. FIRB-Perf 2005: 115-125

[17] K. Jensen: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use.* Volume 1, Basic Concepts. Monographs in Theoretical Computer Science , 1997.

[18] J. Kennedy, and R.C Eberhart : Particle swarm optimization. In: Proc. IEEE Intl.Conf. on Neural Networks, pp. 1942–1948 (1995)

[19] D.S. Kirkpatrick, Jr. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671-680, 1983.

[20] Kelner, V., Capitanescu, F., Léonard, O., and Wehenkel, L. 2008 A hybrid optimization technique coupling an evolutionary and a local search algorithm. *J. Comput. Appl. Math.*

[21] A.Aamodt and E.Plaza, "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches," *AIC* (1994), 39-52.

[22] M. Andries, G. Engels, A. Habel, B. Hoffmann, H.-J. Kreowski, S. Kuske, D. Kuske, D. Plump, A. Schürr, and G. Taentzer. Graph Transformation for Specification and Programming. Technical Report 7/96, Universität Bremen, 1996

[23] D. H. Akehurst and S.Kent. A Relational Approach to Defining Transformations in a Metamodel. *UML 2002* Proceedings, LNCS 2460, 243-258, 2002

[24] Interactive Objects Software GmbH, Project Technology, Inc. MOF 2.0 Query/Views/Transformations RFP, Revised Submission.

[25] Egyed, A.: Heterogeneous Views Integration and its Automation, Ph.D. Thesis (2000)

[26] I. Galvão and A. Goknil, “Survey of Traceability Approaches in Model-Driven Engineering”. *EDOC'07,* pages 313-326,

[27] Jouault, F.: Loosely coupled traceability for atl. In: (ECMDA). (2005)

[28] Marvie, R.: A transformation composition framework for model driven engineering. Technical Report LIFL-2004-10, LIFL (2004)

[29] Adel Ouardani, Philippe Esteban, Mario Paludetto, Jean-Claude Pascal, “A Meta-modeling Approach for Sequence Diagrams to Petri Nets Transformation”, *ESMC* 2006.

[30] M. Wimmer, M. Strommer, H. Kargl, and G. Kramler.Towards model transformation generation by-example. HICSS-40 Hawaii International Conference on System Sciences.

[31] M. Harman and B. F. Jones, Search-based software engineering, *Information & Software Technology*, Vol. 43, No. 14, pp. 833-839 (2001).

[32] O. Seng, J. Stammel, and D. Burkhart. Search-based determination of refactorings for improving the class structure of object-oriented systems. *GECCO '06*, pages 1909-1916..

[33] M. Harman, The Current State and Future of Search Based Software Engineering, In *Proceedings of ICSE 2007*, 20-26 May, Minneapolis, USA (2007)

[34] Yu Sun, Jules White, and Jeff Gray, "Model Transformation by Demonstration," *MoDELS09*, Denver, CO, October 2009, pp. 712-726.

[35] Uzam, M. The use of Petri net reduction approach for an optimal deadlock prevention policy for flexible manufacturing systems. *Int. J. Adv. Manuf. Technol.*, 23, 204–219.

[36] Z. Hu and S. M. Shatz, “Mapping UML Diagrams to a Petri Net Notation for System Simulation”, (SEKE), Banff, Canada, June 2004, pp. 213-219

[37] S. Bernardi, S. Donatelli, J. Merseguer, From UML Sequence Diagrams and StateCharts to analysable Petri Net models, WOSP02, pages 35-45, Rome (Italy), July 2002