Using NSGA-II to compose model transformations alternatives

Smail Rahmoun*†, Etienne Borde†, Laurent Pautet†,
*Institute for Technological Research SystemX - 8, Avenue de la Vauve 91120 PALAISEAU
firstname.lastname@irt-systemx.fr
†Institut Telecom;TELECOM ParisTech; LTCI - UMR 5141 - 46 rue Barrault, 75013 Paris, France
firstname.lastname@telecom-paristech.fr

***Abstract*—The size and complexity of modern systems is rapidly increasing. Meanwhile, the ability to understand and maintain such systems is decreasing almost as fast. Model Driven Engineering promotes the automation of design activities by means of model transformations, enabling to cope with systems complexity. However, the design space produced by the composition of these model transformations is difficult to explore automatically. The main reason that makes this exploration difficult is the size of the design space: the number of possible design choices grows exponentially with the number of model transformation alternatives. This paper presents a Model Driven Engineering approach for design space exploration to deal with the ever-growing challenge of designing complex embedded systems. This approach allows the system architects to automatically select the most adequate model transformations compositions by using multi-objectives optimization evolutionary algorithms.**

I. Introduction

Nowadays, modern systems become more and more large and complex and therefore more difficult to develop and maintain. For example, embedded critical systems such as avionic, railway and automotive are often built to guarantee the safety and robustness properties. To meet these challenging properties makes the design of embedded systems very complex.

Model Driven Engineering (MDE) proposes the use of models as the key assets in the development of such systems. These models cover the whole system development cycle. They can be manipulated manually or automatically using model transformations. These transformations aims to enhance portability by way of separating system (abstract) architecture from platform (concrete) architecture. They facilitate the development of a system in abstraction, and simplifies implementation of that system across a variety of target platforms. In this work, we use model transformations to model the system design alternatives. This process will help system architects to predict Non-Functional Properties (NFP) of a system before its development [2]. This prediction can be exploited to drive decisions about which components should be used and how they should be connected so as to meet the NFP imposed on the design.

Moreover, an issue in modeling a system is that, the NFP could conflict with each other: improving one property can have a negative impact on other properties. For example, improving the availability of a service is often done by replication of the service, which causes the increasing of the response time of these services [19]. Developing a system that satisfies at best all its NFP simultaneously is often not possible: improving a NFP of a system can deteriorate another NFP. As a consequence, system architects have to come up with several design alternatives and select the solutions which fulfils at best all NFP [5] simultaneously from all feasible alternatives. The selected solutions are called non-dominated solutions or Pareto optimal solutions: a solution x2 is non-dominated by another solution x1, if x2 matches or exceeds x1 in all objectives (NFP).

In order to obtain the set of Pareto optimal alternatives, some advanced multi-objective optimization evolutionary algorithms (MOEA) have been proposed. These algorithms apply some fitness evaluations and generic operators. Thus, they are applicable to find solutions in complex design spaces taking into account domain specific characteristics of the problem (e.g. prediction of NFP). MOEA also provides selection operators which guide the search towards interesting regions of the search space by (i) favorizing non-dominated solutions; and (ii) finding a diverse set of solutions in the Pareto front : represents the set of possible solutions.

In this paper, we introduce an approach, tailored for embedded system architecture design, where architecture alternatives are implemented as model transformations driven by NFPs. In order to select good transformations alternatives that help system architects perform trade-off analysis, we apply approaches based on MOEA. More specifically, we map model transformations mechanisms into a multiple-objective optimization problem.

The reminder of this paper is organized as follows. Section II is dedicated to the problem statement. Section III gives an overview of the current advanced MOEA. While Section IV details the approach of this paper. Section V presents the related work.

II. Problem

During the design of a safety-critical system, it is useful to model non functional properties (NFP), such as reliability, availability or temporal performances, already in early stages of the development process. Early NFP modeling enables system designers to make design decisions based on analyses and simulations. But, one major difficulty is that these NFP can be in conflict with each other, that is, improving one NFP can have a negative impact on some others. To construct a system that fulfils all its NFP simultaneously is often not possible. As a consequence, system architects have to consider several design alternatives, and identify a solution that fulfils most NFP from all feasible alternatives.

A possible solution to this issue is to implement these design alternatives in a way that they can be seen as design patterns. Each pattern implements a well-known architectural solution which fulfils a set of NFP. To formalize these design patterns in a way that allows their reuse in different projects or products we implement them as executable model transformations.

The second challenge in this work comes from the size of the design space to explore in order to study such NFP trade-offs. With a complex system and a larger design space, it is almost impossible for software architects to find optimal architecture designs. The exploration of such design space can be very difficult to do manually and error-prone. To overcome this issue, we propose an approach to assist architects throughout the modeling phase and automate the design space exploration by using advanced MOEA.

Moreover, using MOEA to deal with mode transformations alternatives can be challenging. We have (i) to generalize the architecture design problem as a multiple objective optimization problem; (ii) select the more suitable evolutionary algorithm to solve this optimization problem; and (iii) map the notions of model transformation techniques to this evolutionary algorithm.

III. Multi-Objective Evolutionary Algorithm

An evolutionary algorithm (EA) is characterized by three features: (i) a set of solution candidates is maintained; (ii) a mating selection process is performed on this set, and (iii) several solutions may be combined in terms of recombination to generate new solutions.

The solution candidates are called **genomes** (or individuals) and the set of solution candidates is called the **population**. Each genome is composed of a sequence of **genes** and represents a possible solution to the optimization problem.

The mating selection process usually consists of two steps: **fitness evaluation** and **mating pool**. In the first step, the genomes are evaluated in the search space and then assigned a scalar value called **fitness**, reflecting their quality. Afterwards, a **mating pool** is created by random **sampling** from the population according to the fitness values. For example a common used sampling method is **binary tournament selection**. Here two genomes are randomly chosen from the population, and the one with the better fitness value is copied to the mating pool. This procedure is repeated until the mating pool is filled. Then, new solutions are produced by using **genetic operators**: crossover (or recombination) and mutation operators.

Based on the above concepts, EAs are simulated by an iterative computation process. First, an initial population is created at random, which is the starting point of the evolution process. Then a loop consisting of the fitness evaluation and genetic operators is executed a certain number of times. Each loop iteration is called a generation, and often a predefined maximum number of generations serves as the termination criterion of the loop.

In order to deal with the multi-objectives nature of our problem (which model transformations achieve the best possible trade-off among NFP) we use MOEA. Among the many MOEAs that have been proposed in the literature : NSGA-II [6], SPEA2 [20], IBEA [21], only one has been considered in this work, namely NSGA-II (Non-dominated Sorting Genetic Algorithm). It was proposed in the early 90's and still considered today as one of the state-of-the-art MOEA for its robustness across a variety of application domain.

A. *NSGA-II*

NSGA-II is an improved version of NSGA [17] (Non-dominated sorting genetic algorithm). It inherited the non-dominance concept from NSGA and showed two main innovations: non-dominated ranking and crowding distance. Non-dominated ranking is a way to sort genomes in non-dominated fronts whereas crowding distance is a parameter that permits to preserve diversity among solutions of the same non-dominated front: a front represent a set of possible solutions.

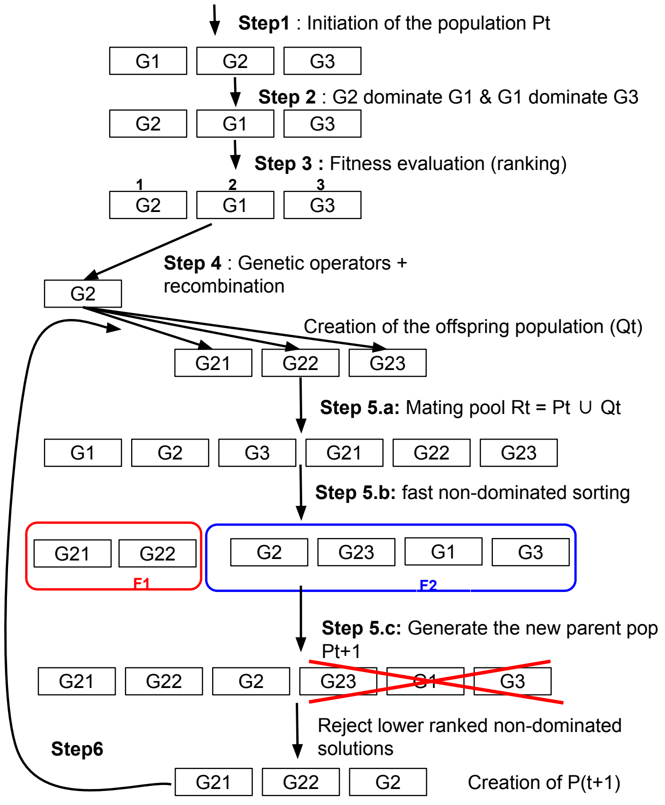

Step1 : Initiation of the population Pt
G1 G2 G3
Step 2 : G2 dominate G1 & G1 dominate G3
G2 G1 G3
Step 3 : Fitness evaluation (ranking)
1 2 3
G2 G1 G3
Step 4 : Genetic operators + recombination
G2
Creation of the offspring population (Qt)
G21 G22 G23
Step 5.a: Mating pool Rt = Pt ∪ Qt
G1 G2 G3 G21 G22 G23
Step 5.b: fast non-dominated sorting
G21 G22
F1
G2 G23 G1 G3
F2
Step 5.c: Generate the new parent pop Pt+1
G21 G22 G2 G23 G1 G3
Reject lower ranked non-dominated solutions
Step6
G21 G22 G2
Creation of P(t+1)

Fig. 1. Schema of the NSGA-II algorithm

Figure 1 gives an overview of how to use NSGA-II to automate the exploration of complex systems design space. As can be seen in the Figure, the algorithm is as follow :

Step 1: Initially, Create a random parent population P_0 composed of N genomes (solutions candidates) (N = 3 in Figure 1). N represents the size of the population.

Step 2: Sort the current population based on a non domination criterion (e.g. reliability, availability and response time).

Step 3: For each non-dominated solution, assign a fitness (rank) equal to its non-domination level (1 being the best level).

Step 4: Create a child population Q_t of size N using binary tournament selection and genetic operators.

Step 5: From the current population, create new generations by following the next operations:

Step 5.1: Create the mating pool R_t of size 2*N by combining the parent population P_t and the child population Q_t.

Step 5.2: Sort the combined population R_t according to the fast non-dominated sorting procedure [6] to identify all non-dominated fronts $(F_1, F_2, ..., F_k)$. Where each front represent a set of possible solutions.

Step 5.3: Generate the new parent population P_{t+1} of size N. The solutions belonging to the best non-dominated front, (i.e. F_1) represent the best solutions in R_t and must be emphasized more than any other solution. If the size of F_1 is smaller than N, all solutions of F_1 are inserted in P_{t+1}. Then, the remaining population of P_{t+1} is chosen from subsequent non-dominated fronts in order of their ranking: the solutions of F_2 are chosen next. However, as shown in Figure 1, not all the solutions from F_2 can be inserted in population P_{t+1}. Indeed, the number of empty slots of P_{t+1} is smaller than the number of fronts. In order to choose which ones will be selected, these solutions are sorted according to their crowding distance and, then, the number of best of them needed to fill the empty slots of P_{t+1} are inserted and the others are rejected.

Step 6: The created population P_{t+1} is then used for selection, crossover and mutation (see Figure 1) to create a new population Q_{t+1}, and so on for the next generations until a stopping criterion is met (e.g. maximum number of generations is reached).

IV. Using evolutionary algorithms for identifying best design alternatives

Figure 2 graphically illustrates the approach proposed in this paper by showing the process underlying it. As can be seen in Figure 2, the process starts with an initial input software architecture which could be designed by some system architect by using architecture design languages. After that, the input model is transformed into a set of alternative architecture models (more detailed models) driven by a set of competing NFP. To solve the trade-off between NFP we expressed the architecture alternatives as design patterns implemented by means of model transformations and compositions techniques. The next step consists at finding the best model transformations alternatives. To do so we (i) map these model transformations alternatives as a multiple-objectives optimization problem and (ii) use NSGA-II to find the non-dominated solutions.

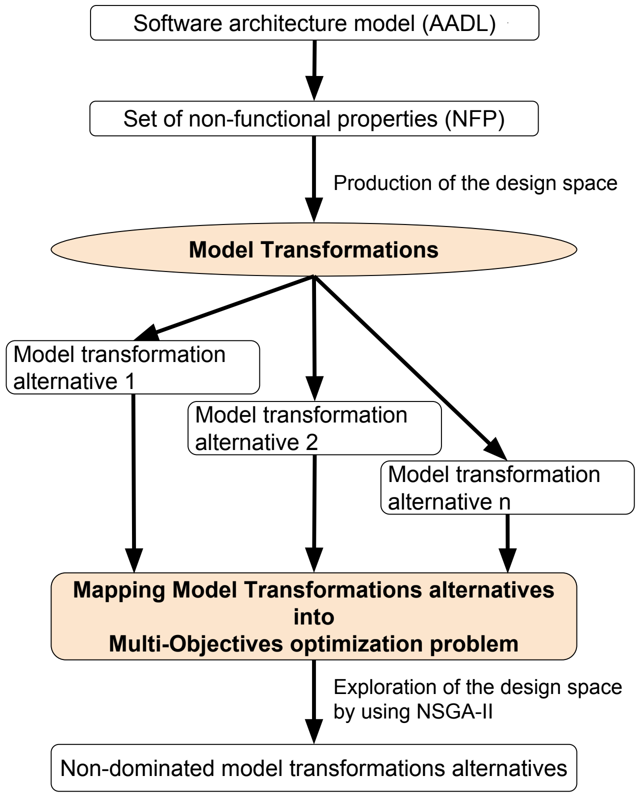

Software architecture model (AADL)
Set of non-functional properties (NFP)
Production of the design space
Model Transformations
Model transformation alternative 1
Model transformation alternative 2
Model transformation alternative n
Mapping Model Transformations alternatives into Multi-Objectives optimization problem
Exploration of the design space by using NSGA-II
Non-dominated model transformations alternatives

Fig. 2. Approach overview

A. System architecture modelisation

Since our application domain is embedded systems, we have chosen AADL (Architecture Analysis & Design Language) as the underlying architecture description language. AADL has been designed on the foundation of the architecture description language MetaH and its goals are (i) specification of quality analysis (e.g. safety); and (ii) design architectures for complex embedded systems. It offers a standardized semantics, mostly expressed using natural language and the components are composed hierarchically according to standardized composition rules. AADL also allows the designers to expand model by defining new property sets in their own specific way. The other reason of choosing AADL is to integrate our work in an existing framework RAMSES[1] which automate re-factoring of model transformations to apply design choices [10].

B. Model transformations composition

In major cases, model transformations are grouped into two main categories of model transformation languages : (i) rule-based transformation languages, or (ii) imperative transformation languages [9]. Rule-based transformation languages such as ATL (for ATLAS Transformation Language) are of great interest in our context since they express model transformation logics in a way that is easy to interpret and **adapt** into other

[1]Refinement of AADL Models for the Synthesis of Embdded Systems. http://penelope.enst.fr/aadl/wiki/Projects

methods (in this paper MOEA). It offers a number of rules that define the mapping from source (abstract architecture) elements to target (concrete architecture) elements. In contrary, the adaptation of imperative transformation languages are more difficult and requires a deep understanding of the control flow graph that leads to the execution of one or another transformation.

In the application of a rule based transformation language, a model transformation MT is generally composed of a set of transformation rules (TR_i). Each transformation rule TR_i is defined by a pair $< E_j, R_j >$. Where E_j is the scope of the transformation rule TR_i. And R_j is the set of rules or actions which TR_i executes. The scope of each transformation rule is defined by pattern matching semantics : a transformation rule TR is applied to every element in the model that matches a boolean expression (e.g. the boolean expression "from" in the ATL language). Then, actions consists of a declarative creation and configuration of elements in the output model.

C. Mapping model transformation alternatives into a multiple objectives optimization problem

The next step of our approach consist in using model transformations compositions described in Section IV-B and integrate them into a multiple-objectives optimization problem (MOOP). Where the population to optimize or the design space represent a set of model transformations alternatives and the objectives to fulfils are NFP such as reliability, availability and response time. The integration process is described in the following sections.

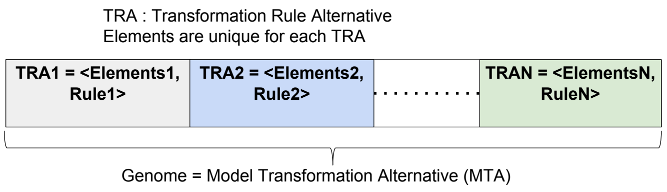

TRA : Transformation Rule Alternative
Elements are unique for each TRA
TRA1 = <Elements1, Rule1>
TRA2 = <Elements2, Rule2>
TRAN = <ElementsN, RuleN>
Genome = Model Transformation Alternative (MTA)

Fig. 3. Genome overview

*1) **Genome representation**:* The first step in a MOOP is to define the population to optimize. In our work, this population represents a set of model transformations alternatives (MTA). Where each genome is a MTA and each gene represent the component of this MTA.

Figure 3 gives an overview of our mapping of a MOOP genome. A genome G represent a MTA and is composed of a sequences of genes. A gene is composed of a set of Transformation Rules Alternative (TRA). Each TRA is, in turn, encoded as a pair $< Elements, Rule >$ where $Elements$ is a list of elements from the input model, and $Rule$ is a transformation rule which can be applied to $Elements$. To be mentioned that, the sequence of transformation rules alternatives is ordered. A TRA exists only when a different rule may be applied to the same set of elements: $\exists\ i, j\ (Elements_i \cap Elements_j \neq \emptyset)$ encodes the existence of transformation rules alternatives TRA_i or TRA_j.

*2) **Fitness evaluation of each genome**:* In each generation, the fitness of every genome in the population is evaluated. The fitness is usually the value of the objective function in the optimization problem being solved. In our work, one fitness function corresponding to three objectives reliability, availability and response time. The aim of our work is to maximize these three objectives. In NSGA-II, the more fit genomes are stochastically selected from the current population, and each genome is modified (recombined and possibly randomly mutated) to form a new generation.

*3) **Genetic operators**:* In NSGA-II, in each iteration, the N genomes selected from the previous generation are used to create new N genomes using genetic operators. This improves the existing solutions by mixing their genetic compositions (crossover) and/or by creating new compositions (mutation). Before applying the operators, the solutions are selected according to their fitness values, see Figure 1. In our work, binary tournament selection, crossover and mutation operators are used.

Binary tournament selection : This operation consists of choosing some genomes at random in the population, and selecting the fittest two for reproduction. The selection criteria are the rank of the containing front and the crowding distance for solutions within the same front. Several tournaments are run to produce the needed genomes.

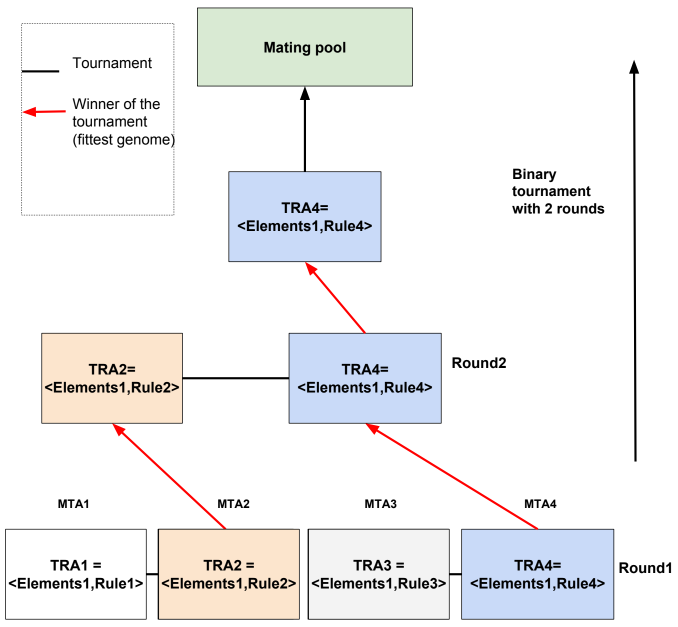
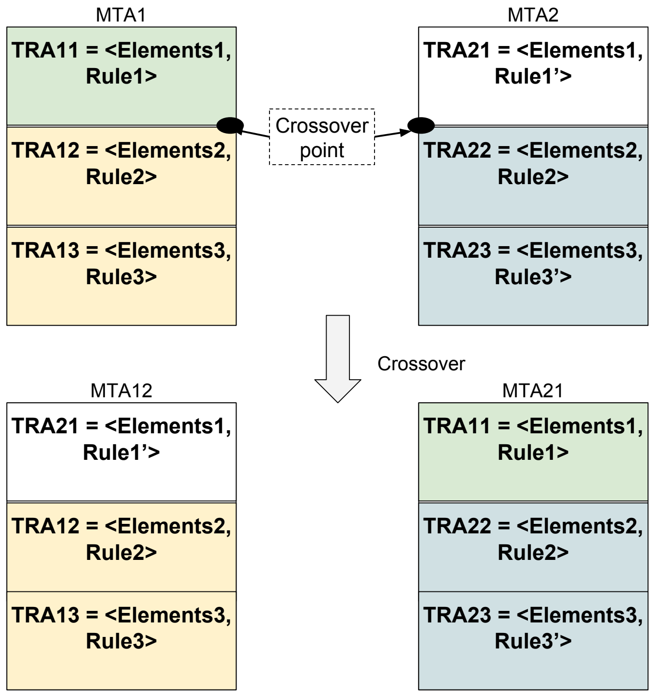


Fig. 4. Tournament Competition between Genomes

Figure 4 illustrates this operation. On this figure, we suppose we have four model transformations alternatives (genomes) MTA1, MTA2, MTA3 and MTA4 respectively composed of the following genes TRA11, TRA21, TRA31 and TRA41. Each TRA is applied to the same set of elements $Elements_1$. We apply two tournaments between paired MTAs to produce the best MTA to apply to $Elements_1$. In the first round we have two tournaments between two different pairs of MTAs

(MTA1 with MTA2 and MTA3 with MTA4). In the second round we apply another tournament on the victorious MTAs in the previous round (MTA2 and MTA4) to get the fittest MTA which will be put in the mating pool for reproduction (generate new MTAs). In Figure 4 the fittest MTA which is expected to appear in the mating pool is MTA4.

Crossover : The crossover consists in producing new genomes from the existing ones. When two genomes are selected using the binary tournament selection, two offspring solutions are created, with a given crossover probability, by exchanging parts of the parent genomes. This consists in randomly selecting a cut point in the genome, and all the genes beyond that point in either parent are swapped between the two parents.

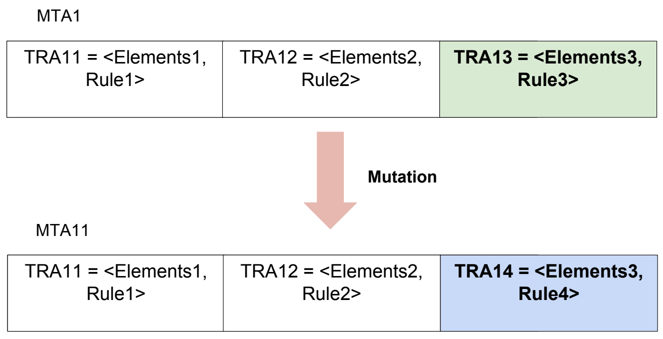


MTA1
MTA2
TRA11 = <Elements1, Rule1>
TRA21 = <Elements1, Rule1'>
Crossover point
TRA12 = <Elements2, Rule2>
TRA22 = <Elements2, Rule2>
TRA13 = <Elements3, Rule3>
TRA23 = <Elements3, Rule3'>
Crossover
MTA12
MTA21
TRA21 = <Elements1, Rule1'>
TRA11 = <Elements1, Rule1>
TRA12 = <Elements2, Rule2>
TRA22 = <Elements2, Rule2>
TRA13 = <Elements3, Rule3>
TRA23 = <Elements3, Rule3'>

Fig. 5. Crossover operator

Figure 5 illustrates the result of applying the crossover operator to two parent genomes MTA1 and MTA2 respectively composed of the following genes {TRA11, TRA12, TRA13} and {TRA21, TRA22, TRA23} to produce new genomes MTA12 and MTA21, respectively composed of new sequences {TRA21,TRA12,TRA13} and {TRA11,TRA22,TRA23}. This operator combines MTA1 and MTA2 by selecting a single point on the genome and swapping the genes between the two genomes that lie beyond this point.

Mutation : After performing the crossover, the obtained solutions could be mutated with a given mutation probability. The mutation operator acts on a single genome MTA1 to obtained a muted genome MTA11 acts as follows. The mutation operator selects a gene at random in the genome and changes its value. In Figure 6, the third gene TRA13 is selected to mutate into a new gene TRA14 : we apply a different rule (i.e. rule4) to the set of selected elements.



MTA1
TRA11 = <Elements1, Rule1>
TRA12 = <Elements2, Rule2>
TRA13 = <Elements3, Rule3>
Mutation
MTA11
TRA11 = <Elements1, Rule1>
TRA12 = <Elements2, Rule2>
TRA14 = <Elements3, Rule4>

Fig. 6. Mutation operator

The final step of our approach consists in using the NSGA-II algorithm described in Section III to our optimization design problem defined above to find the set of non-dominated model transformations alternatives.

V. Related Work

Our work is based on system performance prediction [2] by using model transformations techniques and multi-objective metaheuristic optimization [5].

A. model transformations techniques

Recently, some works have focused on the definition of model transformation alternatives. The first works introduced the superimposition mechanism [16] in order to to implement transformations alternatives by replacing parts of a rule-based model transformation. Implementing alternatives while reusing rules was also an important challenge studied in [13].

These works are the building blocks of other works, such in [8] where they performed model transformations factorizations and compositions in order to implement quality driven model transformation alternatives : specification of some variants in the source code of model transformations in order to produce output models having different quality attributes.

In [10] the authors implement a framework that improve system architectures by using a semi-automatic selection and specialization of model transformation alternatives while taking care of transformation dependenciess.

Our contribution is based on these work since they enable the factorization and composition of model transformations. However, these approaches don't propose any algorithm to select the non-dominated (or pareto) model transformations alternatives which fulfil at best a set of competing non-functional properties.

B. multi-objective metaheuristic optimization

For the optimization of NFP, the first works consisted at using a cyclic process [3]. The benefit of this process is its simplicity. The process can be tailored to different non-functional properties by using specific evaluation methods and architecture transformations. But this process focuses on only one non-functional property at a time. Unlike in our work, we take into account several properties at a same time (e.g. reliability, availability and response time).

Aleti et al. [1] have developed the ArchOpteryx framework. ArchOpteryx assists the system architects during the design phase to create architectures that meet non-functional properties by using multi-objective optimization strategies.

Koziolek et al. [15] also developed their own framework Peropteryx for improvement of system architecture models using a metaheuristic search guided by architectural tactics.

Unlike these two frameworks, in our approach design alternatives are implemented as model transformations driven by NFP. The use of model transformations help the design architects to (i) automating the system production; (ii) reusing architecture alternatives in other projects (or products) and (iii) solving the competition issue between NFP.

VI. Conclusion

In this paper we propose an approach which aims at improving system architecture models using two mechanisms: (ii) using model transformations compositions to formalize model design alternatives; and (ii) mapping these model transformations to an elitist multiple-objective optimisation algorithm to explore the design space.

The main benefits of this approach is to handle a large search spaces of quality optimization problems and to automate the identification of good architecture design alternatives. This will reduce development costs and improve the quality of the final system. Additionally this approach enables the ranking of design alternatives with respect to conflicting NFP and the possibility to reuse these alternatives in other projects or products by reusing model transformations.

In the future, we plan to integrate our approach in the framework RAMSES and evaluating our approach in more details quantitatively (i.e. experimenting it on a an industrial case study) [4].

VII. Acknowledgment

This research work has been carried out under the leadership of the Technological Research Institute SystemX, and therefore granted with public funds within the scope of the French Program “Investissements d’Avenir”.

References

[1] A. Aleti, S. Bjornander, L. Grunske, and I. Meedeniya, “Archeopterix: An extendable tool for architecture optimization of aadl models,” in *Model-Based Methodologies for Pervasive and Embedded Software, 2009. MOMPES ’09. ICSE Workshop on*, May 2009, pp. 61–71.

[2] S. Balsamo, A. di Marco, P. Inverardi, and M. Simeoni, “Model-based performance prediction in software development: a survey,” *Software Engineering, IEEE Transactions on*, vol. 30, no. 5, pp. 295–310, May 2004.

[3] J. Bosch, P. Molin, J. Bosch, and P. Molin, “Software architecture design: Evaluation and transformation,” in *In 1999 IEEE Engineering of Computer Based Systems Symposium. IEEE Computer Based Systems*, 1999, pp. 4–10.

[4] F. Cadoret, T. Robert, E. Borde, L. Pautet, and F. Singhoff, “Deterministic implementation of periodic-delayed communications and experimentation in aadl,” Paderborn, Allemagne, June 2013.

category=inproceedings language=en audience=2 state=toappear project=safir dept=infres group=s3 id=13716

[5] C. A. C. Coello, “A comprehensive survey of evolutionary-based multiobjective optimization techniques,” *Knowledge and Information Systems*, vol. 1, pp. 269–308, 1998.

[6] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *Trans. Evol. Comp*, vol. 6, no. 2, pp. 182–197, Apr. 2002. [Online]. Available: http://dx.doi.org/10.1109/4235.996017

[7] K. Deb and D. Kalyanmoy, *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: John Wiley &amp; Sons, Inc., 2001.

[8] M. Drago, C. Ghezzi, and R. Mirandola, “Towards quality driven exploration of model transformation spaces,” in *Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science, J. Whittle, T. Clark, and T. Khne, Eds., vol. 6981. Springer Berlin Heidelberg, 2011, pp. 2–16. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24485-8_2

[9] P. V. Gorp, O. Muliawan, D. Janssens, P. V. Gorp, O. Muliawan, and D. Janssens, “Integrating a declarative with an imperative model transformation language,” 2006.

[10] G. Loniewski, E. Borde, D. Blouin and E.Insfran ´, “An Automated Approach for Architectural Model Transformations,” in *Information Systems Development, 22nd International Conference, ISD 2013, Sevilla, Spain, September 2013. Proceedings*, 2013, pp. 295-306. [Online]. Available: http://infres.enst.fr/~borde/isd2013loniewski.pdf

[11] A. Johnsen and K. Lundqvist, “Developing dependable software-intensive systems: Aadl vs. east-adl,” in *Ada-Europe 2011*, A. Romanovsky and T. Vardanega, Eds. Springer-Verlag, June 2011, pp. 103–117. [Online]. Available: http://www.es.mdh.se/publications/1753-

[12] F. Jouault and I. Kurtev, “Transforming models with atl,” in *Proceedings of the 2005 International Conference on Satellite Events at the MoDELS*, ser. MoDELS’05. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 128–138. [Online]. Available: http://dx.doi.org/10.1007/11663430_14

[13] A. Kavimandan, A. Gokhale, G. Karsai, and J. Gray, “Managing the quality of software product line architectures through reusable model transformations,” in *Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS*, ser. QoSA-ISARCS ’11. New York, NY, USA: ACM, 2011, pp. 13–22. [Online]. Available: http://doi.acm.org/10.1145/2000259.2000264

[14] A. G. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[15] A. Koziolek, H. Koziolek, and R. Reussner, “Peropteryx: Automated application of tactics in multi-objective software architecture optimization,” in *Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS*, ser. QoSA-ISARCS ’11. New York, NY, USA: ACM, 2011, pp. 33–42. [Online]. Available: http://doi.acm.org/10.1145/2000259.2000267

[16] E. Navarro, C. E. Cuesta, D. E. Perry, and C. Roda, “Using model transformation techniques for the superimposition of architectural styles,” in *Proceedings of the 5th European Conference on Software Architecture*, ser. ECSA’11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 379–387. [Online]. Available: http://dl.acm.org/citation.cfm?id=2041790.2041840

[17] N. Srinivas and K. Deb, “Multiobjective optimization using nondominated sorting in genetic algorithms,” *Evolutionary Computation*, vol. 2, pp. 221–248, 1994.

[18] M. Tisi, F. Jouault, P. Fraternali, S. Ceri, and J. Bézivin, “On the use of higher-order model transformations,” in *Model Driven Architecture - Foundations and Applications, 5th European Conference, ECMDA-FA 2009, Enschede, The Netherlands, June 23-26, 2009. Proceedings*, 2009, pp. 18–33. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02674-4_3

[19] H. Yu and A. Vahdat, “The costs and limits of availability for replicated services,” *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 29–42, Oct. 2001. [Online]. Available: http://doi.acm.org/10.1145/502059.502038

[20] E. Zitzler, M. Laumanns, and L. Thiele, “Spea2: Improving the strength pareto evolutionary algorithm,” Tech. Rep., 2001.

[21] E. Zitzler and S. Knzli, “Indicator-based selection in multiobjective search,” in *in Proc. 22nd International Conference on Information Systems Development (ISD 2013*. Springer, 2004, pp. 832–842.