Using NSGA-II to compose model transformations alternatives

Smail Rahmoun
Institute for Technological Research
SystemX
Palaiseau France
Email: smail.rahmoun@irt-systemx.fr

Etienne Borde
and Laurent Pautet
TELECOM Paristech Institut
LTCI-UMR 5141
Paris, France
Email: firstname.name@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 (MDE) 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 make 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 MDA 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 (MOEA) mechanisms.**

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 which are often built to accomplish specific functionalities, can integrate thousands of buses, software and hardware components. To guarantee properties such as safety, availability and robustness makes the design of these systems very challenging and complicated.

Model Driven Engineering (MDE) proposes the use of models as the key assets in the development of such systems, and hence all sorts of model modifications are needed. In this way, model transformations become one of the basic building blocks of MDE. Even though MDE is being successfully applied in many scenarios, it still needs appropriate mechanisms to handle the development of complex, large-scale systems. One such mechanism is a facility to make transformations reusable, so that we can apply them to different contexts. In this work, we use generic model transformation to model design alternatives of the system. This process will help us to predict NFP of a system before it 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 the NFP of a system is that, the NFP could conflict[1] with each other. 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 [20]. 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 optimal solution (Pareto optimal solutions) from all feasible alternatives through the trade-off analysis regarding all NFP [4].

In order to obtain the set of Pareto optimal alternatives, some advanced multi-objective optimization evolutionary algorithms (MOEA) have been proposed. These algorithm apply some fitness evaluations and generic operators of variation. Thus, they are applicable to search spaces (e.g. design space) 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 : where a solution x1 is dominated by another solution x2 , if x2 matches or exceeds x1 in all objectives; (ii) finding a diverse set of solutions on the Pareto front.

Our approach, consists in improving the NFP of an architecture design through a generic method using model transformations compositions and advanced MOEA. This method will automate the search for optimal compositions from all feasible alternatives. More specific, beginning with some initial architecture that specifies a set of NFP, design alternatives are generated by applying model transformation compositions [12]. To get the non-dominated alternatives, model transformation rules enter into a meta-heuristic loop. That is, we generate transformation rules alternatives (TRA) through some genetic operations such as mutation and crossover. The models transformed by the selected TRA are evaluated and corresponding multiple NFP criteria of interests are obtained after application of transformations in order to obtain more precise analysis.

The reminder of this paper is organized as follows. Section II is dedicated to the problem statement. Section III gives an overview of our approach. While Section IV details our

[1] improving one property can have a negative impact on other properties

approach. 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 the developer of the system to make design decision 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 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 (an optimal solution) 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. Where each pattern implement a well-known architectural solution which fulfils a set of NFP. To formalize these design patterns in a way that allow their reuse in different projects or products we implement them as 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 existing multiple objective evolutionary algorithms (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) adapting the notions of model transformation compositions to this evolutionary algorithm.

III. Using evolutionary algorithms for identifying good design alternatives

Figure 1 graphically illustrates the approach proposed in this paper by showing the process underlying it. As can be seen in Figure 1, the process starts with an initial input software architecture which could be designed by software architects by using some architecture design language [16]. After that, this architecture model is transformed into a set of alternative architecture models (more detailed models) which fulfils a set of competing NFP such as reliability, availability and response time. 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 final step in Figure 1 consists in exploring the design space composed of all model transformation compositions and select which transformation fulfil at best the NFP of the system. This step is done by using an advanced evolutionary algorithm.

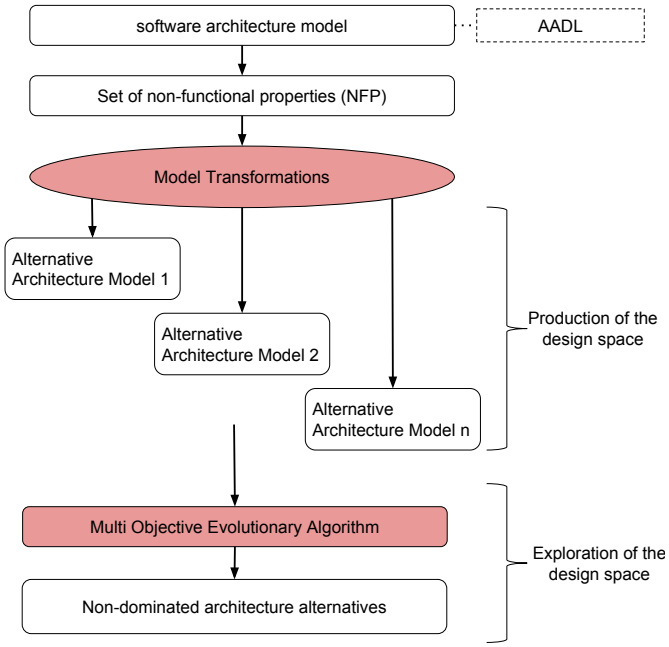

Fig. 1. Overview of the identification step

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[2] which apply selection of design alternatives.

B. Multi-Objective Genetic Algorithm

In order to deal with the multi-objective nature of our problem (which model transformation achieve the best possible trade-off among NFP) we use MOEA. Among the many MOEAs that have been proposed in the literature : NSGA-II [5], SPEA2 [21], IBEA [22], only one have 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.

C. NSGA-II

NSGA-II is an improved version of NSGA [18] (Non-dominated sorting genetic algorithm). It inherited the non-

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

dominance concept from NSGA and showed three innovations, (i) a crowding approach for diversity preservation; (ii) an elitism operator that helps in significantly speeding up the performance of the genetic algorithm (fast non-dominated sort); and (iii) an objective-wise distance computation.

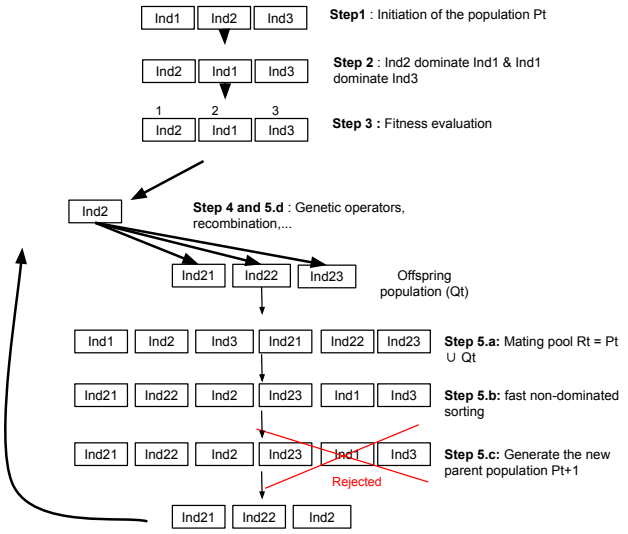

Fig. 2. Schema of the NSGA-II algorithm

Figure 2 gives an overview of how to use NSGA-II to automate the exploration of complex systems design space. The steps of this algorithm are as follow :

1) Initially, Create a random parent population P_0 composed of N genomes (N = 3 in Figure 2).
2) Sort the current population based on a non domination criterion (e.g. reliability, availability and response time).
3) For each non-dominated solution, assign a fitness (rank) equal to its non-domination level (1 is the best level, 2 is the next best level, and so on).
4) Create a child population Q_0 of size N using binary tournament selection, recombination, and mutation operators.
5) From the first generation onwards, creation of each new generation constitutes the following steps :
   a) Create the mating pool R_t of size 2N by combining the parent population P_t and the child population Q_t.
   b) Sort the combined population R_t according to the fast non-dominated sorting procedure to identify all non-dominated fronts $(F_1, F_2, ..., F_k)$.
   c) Generate the new parent population $P_t + 1$ of size N by adding non-dominated solutions starting from the first ranked non-dominated front $F1$ and proceeding with the subsequently ranked non-dominated fronts $F_1, F_2, ..., F_k$, till the size exceeds N (Figure 2). This means that the total count of the non-dominated solutions from the fronts $F_1, F_2, ..., F_k$, exceeds the population size N. Now, in order to make the total count of the non-dominated solutions equal to N, it is required to **reject** some of the lower ranked non-dominated solutions from the last $Fkth$ front. This is achieved through a sorting process based on the crowding distance assigned to each solution contained in the F_kth non-dominated front. Thus, the new parent population $P_t + 1$ of size N is constructed.
   d) Perform the selection, crossover and mutation operations on the newly generated parent population (P_t+1) to create the new child population $(Q_t+1))$ of size N (Figure 2).
6) Repeat Step 5 until a stopping criterion is met (e.g. maximum number of generations is reached).

IV. Compose model transformation alternatives by using NSGA-II

In this paper we propose a generic method to automate the search of optimal architectures (which fulfil at best the NFP of the system) from the design space. To represent the design space (set of architecture alternatives) we use model transformations compositions. The exploration step is done by using the NSGA-II algorithm.

A. Model transformations languages

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].

The first category express model transformation logics in a way that is easy to interpret and **adapt**. It permit to modify the pattern matching part of a transformation rule TR which is sufficient to modify the set of elements this rule applies to. 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.

We used in our work the first category of languages since it more easy to adapt in other methods (in this paper MOEA). Where model transformations are endogenous : the source and target models are the same. In our work we use a AADL to AADL model transformations in order to refine or improve the architecture of an embedded system while representing the result of this refinement in an analyzable model. More specific, we use the rule-based transformation language ATL (for Atlas Transformation Language).

A rule-based transformation T is generally composed of a set of transformation rules $(TR_j, j \in 1, 2, .., n)$. Each transformation rule TR_j is defined by a pair $< E_j, R_j >$, where E_j is the scope of the transformation rule TR_j : the set of elements from the input model the transformation rule TR_j can be applied to. And R_j is the set of rules or actions which TR_j executes when it is applied to an element of E_j (i.e. creation of elements in the target model).

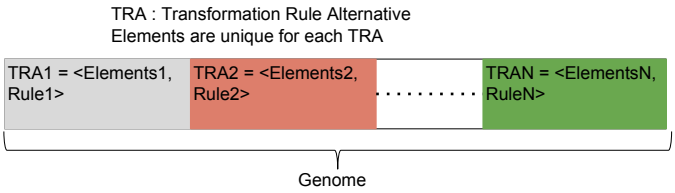

Fig. 3. Genome overview

B. Genome representation

The first step of our work consists in applying the definition of the multi-objective optimization problem on our architecture design problem. Where the population to optimize or the design space represent a set of architecture alternatives and the objectives functions are f_1 : reliability, f_2 : availability and f_3 : response time. The architecture alternatives are formalized in a way that allow their reuse in different projects or products. To do so, we use model transformations mechanisms. An issue in using such methods is their complexity : how to identify which elements of the input architecture model transformation alternatives can be applied. To ease the development and maintenance of such model transformations, we propose to use a rule-based transformation languages and composition mechanisms.

In our approach we have several Transformation Rules Alternatives (TRA). They represent the genomes to optimize in our multi-objective problem. Figure 3 gives an overview of these genomes. Each genome $G_i, i \in 1, 2, ..., n$ is composed of a set of TRA $(G = TRA_i, i \in 1, 2, ...n)$. Each TRA_i is, in turn, encoded as a pair $TRA_i =< Elements_i, Rule_i >$ where $Elements_i (i = 1, 2, ..., n)$ is a list of elements from the input model, and $Rule_i (i = 1, 2, ..., n)$ is a transformation rule which can be applied to $Elements_i$.

C. Genome evaluation

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. We defined 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.

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.

D. Genetic operators definition

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 material (crossover) and/or by creating new material (mutation). Before applying the operators, the solutions are selected according to their fitness values, see Figure 2. In our work, binary tournament selection, crossover and mutation operators are used.

*1) **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 N needed genomes.

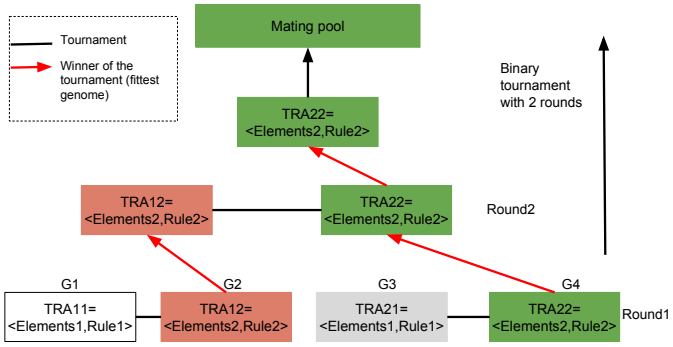

Fig. 4. Tournament Competition between Genomes

Figure 4 illustrates this operation. On this figure, we suppose we have four genomes (G1,G2,G3,G4) and we apply two tournaments to produce the needed genomes. The final genomes of the competition will always be the fittest genomes in the population. The more successful the genome is in competition, the more often that genome is expected to appear in the mating pool (In Figure 4 the fittest genome is G4).

*2) **Crossover**:* The crossover consists of 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 vector, and all the target fragments beyond that point in either parent are swapped between the two parents.

Figure 5 illustrates the result of applying the crossover operator to two parent genomes G1 and G2 respectively composed of {TRA11,TRA12,TRA13} and {TRA21,TRA22,TRA23} to produce a new genome G3. As can be seen in Figure 5, this operator combines G1 and G2 by selecting a single point on the genome and swapping the genes between the two genomes that lie beyond this point.

*3) **Mutation**:* After performing the crossover, the obtained solutions could be mutated with a given mutation probability.

The mutation operator act on a single genome G1 to obtained a muted genome G11 acts as follows. The mutation operator selects a gene at random in the genome and changes its value. In Figure 6, the third tile TRA13 is selected to mutate into TRA14 : we apply a different rule (i.e. rule4) to the set of selected elements.

V. Related Work

Our work is based on system performance prediction [2] by using model transformations techniques and multi-objective metaheuristic optimization [4].

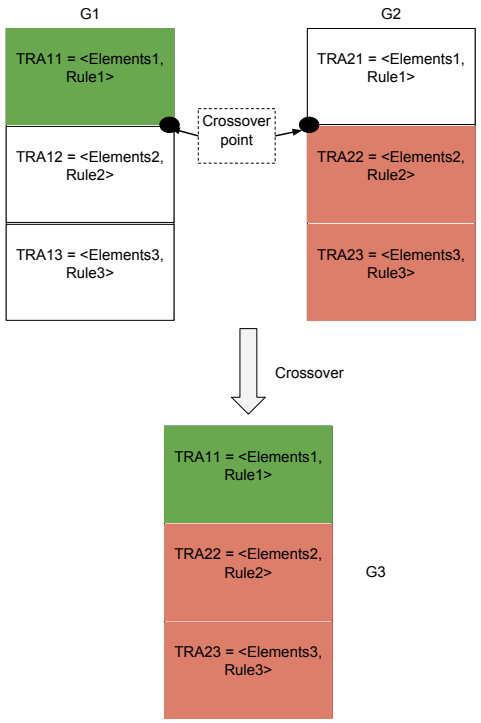

Fig. 5. Crossover operator

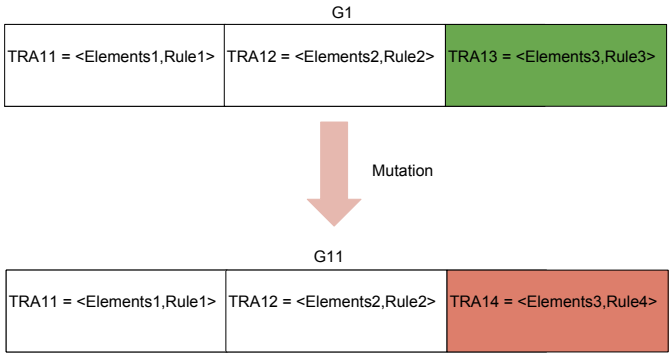

Fig. 6. Mutation operator

A. model transformations techniques

Recently, some works have focused on the definition of model transformation alternatives. The first works introduced the superimposition mechanism [17] to implement transformations alternatives. Another work [13] proposed to implement transformations alternatives by using rule-based transformation languages. These works has been continued in [8] in order to deal with architecture quality problems.

Our contribution is based on this work since it enable the factorization and composition of model transformations. However, this approach doesn'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

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.

Compared to DesignBots [7], in our method model transformations are not decomposed into subplans, each transformation is performed atomically.

For the optimization of NFP another work consists 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 us who take into account several properties at a same time (e.g. reliability, availability and response time).

VI. Conclusion

An architecture design method has been presented that explicitly addresses the NFP put on the architecture. The proposed approach aims at improving system architecture models using two mechanisms. On the one hand, model transformations compositions to formalize model design alternatives. On the other hand, an elitist multiple-objective optimisation algorithm to explore this 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) [10].

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 dAvenir".

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

[5] 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

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

[7] J. A. Díaz-Pace and M. R. Campo, "Using planning techniques to assist quality-driven architectural design exploration," in *Proceedings of the Quality of Software Architectures 3rd International Conference on Software Architectures, Components, and Applications*, ser. QoSA'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 33–52. [Online]. Available: http://dl.acm.org/citation.cfm?id=1784860.1784865

[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] N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Trans. Softw. Eng.*, vol. 26, no. 1, pp. 70–93, Jan. 2000. [Online]. Available: http://dx.doi.org/10.1109/32.825767

[17] 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

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

[19] 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

[20] 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

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

[22] 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.