Towards Model-Based Optimisation: Using Domain Knowledge Explicitly

Steffen Zschaler[1] and Lawrence Mandow[2]

[1] Department of Informatics
King's College London
szschaler@acm.org

[2] Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga
lawrence@lcc.uma.es

Abstract. Search-based software engineering (SBSE) treats software-design problems as search and optimisation problems addressing them by applying automated search and optimisation algorithms. A key concern is the adequate capture and representation of the structure of design problems. Model-driven engineering (MDE) has a strong focus on domain-specific languages (DSLs) which are defined through meta-models, capturing the structure and constraints of a particular domain. There is, thus, a clear argument for combining both techniques to obtain the best of both worlds. Some authors have proposed a number of approaches in recent years, but these have mainly focused on the optimisation of transformations or on the identification of good generic encodings of models for search. In this paper, we first provide a structured overview of the current state of the art before identifying limitations of the key proposals (transformation optimisation and generic genetic encodings of models). We then present a first prototype for running search algorithms directly on models themselves (rather than a separate representation) and derive key research challenges for this approach to model optimisation.

Keywords: evolutionary optimisation, object space, model-driven engineering, model transformations

1 Introduction

Search-based software engineering (SBSE) is about using optimisation techniques for automating the search for (near-)optimal software designs [1]. More recently, the use of search-based approaches has also been extended to software adaptation (e.g., [2,3]). Using search-based techniques allows the exploration of much larger design spaces than could be explored manually by developers. As a result, better solutions can be identified more quickly.

However, as has been recognised before [4,5], the problem domains in software engineering are too complex to be effectively captured with traditional problem representations as they are typically used in search-based systems. Model-driven engineering (MDE) offers good techniques for capturing complex domains including their structural constraints by using meta-models. As a result, there has recently been increased interest in combining the advantages of SBSE and MDE [6,7,8,9,10,11,12,13,14,15].

Much of this work has focused on finding good generic representations of models that are tailored towards the needs of standard optimisation algorithms (most typically, genetic algorithms, e.g. [16]). As we will discuss in detail in Sect. 3, these generic encodings introduce their own challenges. Most importantly, they make it easy for search steps to produce invalid candidate solution; that is, models that do not satisfy the constraints expressed by the meta-model or its well-formedness rules.

We propose an alternative approach: instead of defining a secondary encoding for candidate solutions, we propose to run optimisation algorithms directly on models represented in standard meta-modelling data structures. We argue that given that developers have spent substantial time and effort designing meta-models that are a good representation of the domain, we should make use of as much of this information as possible during search and optimisation. We present a first prototype of a tool for running population-based optimisations directly on models and discuss some of the research challenges that need to be addressed to make this vision a reality.

The remainder of this paper is structured as follows: In the next section, we give an overview of existing work defining generic approaches to SBSE in an MDE context. We then discuss some of the limitations of these existing works in Sect. 3, before presenting our own proposal in Sect. 4. Our work is only an initial exploration, highlighting more problems than providing definitive solutions, so in Sect. 5, we outline some of the key research challenges in this area.

2 Related Work

Table 1 summarises the current literature on optimisation in MDE.[3] We use two orthogonal sets of categories for characterising the different approaches that have been explored so far:

1. *By optimisation target.* Some approaches focus on selecting optimal *transformations* (producing optimal models), while other approaches focus on finding optimal *models* directly without considering optimality of transformations (and, possibly, without explicitly specified transformations).
2. *By encoding approach.* Two general approaches have emerged: (1) general encoding of MDE artefacts using standard genetic encodings and applying genetic optimisation algorithms, and (2) techniques that work directly on the structure of the MDE artefacts themselves (possibly with annotations).

We will briefly discuss these categories in more detail below.

[3] Some other approaches exist, but they have only been defined for a specific problem. Here, we focus on approaches that aim to be generic.

	Optimisation of models	Optimisation of transformations
Genome encoding	[11,10,5]	[6,13]
Direct search	[12,17,4]	[7,8,9,15]

Table 1. Overview of related work

2.1 Optimisation of transformations

A number of authors have proposed approaches that aim to search for optimal transformations instead of looking only for optimal models. Optimality of the models produced is still of interest, but they are only indirectly manipulated by searching the space of viable model transformations. This indirect approach is chosen for one of three reasons:

1. There may be optimality criteria directly on the transformations themselves. For example, when optimising cloud data centre configurations based on a current configuration [3], we are not only interested in finding the optimal new configuration, but also the shortest or least costly path there. Searching for transformations rather than for models directly allows these objectives to be expressed and incorporated into the search process.
2. There might be reachability constraints requiring valid solutions to be reachable from an initial model through a sequence of model transformations. Optimising transformation chains enables these constraints to be expressed.
3. The transformations encode developer choices, but may need to be rerun for other source models at a later stage. In other words, the transformations are the actual optimisation object, their application to models is separate to the optimisation process.

Using genome encodings. Abdeen *et al.* [13] propose encoding different transformation sequences as genomes so that they become amenable to search with a genetic algorithm. Their approach is based on Viatra. Each candidate solution is a tuple made from a start model and a sequence of applications of small, predefined Viatra transformations. The paper provides some good discussion of the issue of invalid individuals: these may be the result of crossover or mutation, making transformation chains invalid or causing them to create invalid models. [13] uses a specific ranking mechanism to handle these cases.

Fleck *et al.* [6], propose MOMoT. MOMoT uses base transformations expressed in Henshin and optimises chains of applications of these transformations to a base model. Fleck *et al.* handle invalid candidate solutions with a repair mechanism, effectively reducing the population size for these situations.

Using direct search. Drago *et al.* [7,8,9] proposed QVT-R^2, an extension of QVT-Relational. In QVT-R^2, transformation developers create non-deterministic transformations by providing multiple rules matching the same structure in the source model. The rules are then annotated with information about relevant quality analysis (e.g., information about how to invoke an external performance analysis). Each choice point is then incrementally and interactively fixed for a given source model by running all transformation options and asking the developer to choose the best resulting model based on the automatically invoked quality analyses.

Hegedüs *et al.* [15] describe an approach to the guided exploration of transformation chains, using a single-state, back-tracking algorithm guided by dependency graphs between the individual transformations. Their work is based on their earlier work on CPS(M) [14], where they introduced an approach to search-based constraint solving directly over models.

2.2 Optimisation of models

Using genome encodings. Williams [10] proposes Crepe, a generic encoding of models as sequences of integers. Given a meta-model and a so-called "finitisation model" (containing information about the range of attributes), Crepe provides a unique, bi-directional transformation between instances of the meta-model and integer-vector representations of these models. The integer vectors can be used as genomes in the context of genetic algorithms using standardised, domain-independent operators for mutation and crossover. In [11], the authors provide some extensions to the approach as well as a first comparative evaluation of its performance compared to a manually implemented optimisation.

Kessentini *et al.* [5], give a high-level overview of an architecture for reusing SBSE techniques in the context of MDE. A key ingredient of their approach is also a generic meta-model for encoding models as genomes, making them amenable to standard genetic operators such as mutation and crossover.

Using direct search. Burton and Poulding [4] were the first to describe an idea for running optimisation directly on models.[4] They create separate domain-specific modelling languages describing a search problem and candidate solutions and run search to find near-optimal solutions. As described in [17], their solutions are mappings between solution and problem models, effectively limiting the problem to the optimisation of vectors of binary associations. This enables them to easily define general mutation and crossover operators so that standard evolutionary search can be applied.

Denil *et al.* [12] present a general approach for model-based optimisation using their Formalism Transformation Graph and Process Model (FTG+PM). Their main focus is implementing different search algorithms as transformation scheduling programs, fully integrating them into a general MDE approach. They show how to implement a number of single-state search algorithms in this way, applying them to a problem in circuit design. Candidate solutions are implicit in the transformation scheduling language, which may make it difficult to extend this approach to population-based search algorithms.

3 Issues with Generic Encoding of Models

We are interested in optimisation of models rather than transformations. While the latter is useful when we have optimality requirements over transformation

[4] In earlier work, Horvath *et al.* [14] introduce the idea of search over models, but without support for optimisation.

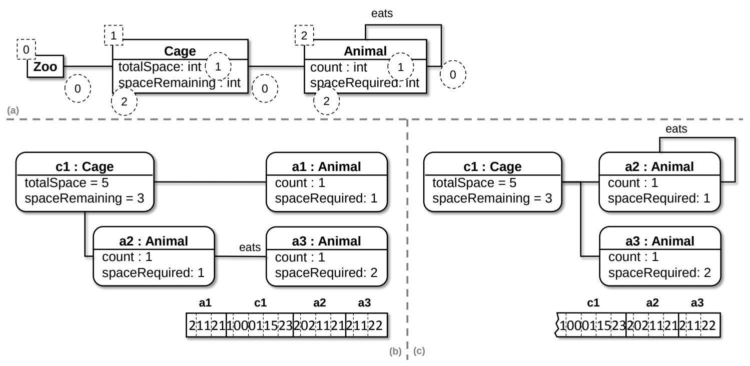


Fig. 1. Zoo DSL example (based on [10])

chains (or reachability constraints expressed through such chains), more typically, we are interested simply in deriving optimal models. In such a case, optimising transformations incurs too much overhead in repair and through redundant representations of the same model through different transformation chains (effectively reducing the size of the search population).

Generic genetic encodings of models as proposed in [5,10,11], however, have their own problems. In particular, it seems very difficult to ensure locality and preservation of well-formedness as we will demonstrate in an example. Figure 1 (a) shows the meta-model of a simple Zoo DSL. This DSL allows the description of zoo configurations, where there are cages with animals, some of which may eat other animals. An optimisation problem of interest would be to find the minimum number of cages to keep all the animals in so that there are no animals that eat each other in the same cage. Note that cages have a maximum capacity and animals require a certain amount of space. The meta-model has been annotated to indicate how it would be encoded by the algorithm proposed in [10]. There, models are encoded as integer strings, where each model element is started by an integer identifying its meta-class (shown in dashed rectangles in Fig. 1 (a)). This is then followed by pairs of integers where the first identifies the structural feature (indicated by dashed circles in the figure) and the second provides the value for this feature. For associations, the values are provided by numbering all instances of the target type in sequence of appearance in the gene.

Figure 1 (b) shows an example model[5] and its encoding as a gene using this algorithm. To ensure smooth applicability of the standard mutation and crossover operators, genetic encodings should be local; that is, changes in one part should not affect other parts of the represented object. This is not the case for this generic encoding. For example, Fig. 1 (c) shows the same gene after it has been split in preparation of a standard single-point crossover operation, removing the representation for object `a1` only. The remainder of the gene is exactly as

[5] We leave out the recurring `Zoo` object for simplicity.

before, but the model fragment encoded by it has changed dramatically: the eat relationship between the two animals has been lost and `a3` has been moved into the cage. Note that the latter change has also led to a violation of the model's well-formedness rules, which require that `spaceRemaining` should always indicate the space left to place additional animals in a cage (and so should now be 2). We are not aware of any generic genetic encoding of models that has overcome this problem using generic mutation and crossover operators. As a result, such approaches require a lot of repair of candidate solutions, substantially worsening the optimisation performance [11].

The problem could be resolved by defining domain-specific mutation and crossover operators. However, these are difficult to implement on the level of genotypes; they will effectively have to constantly reinterpret the genes as the corresponding model fragments. We have explored this in [18], showing that such specific operators do indeed have a positive effect on the performance of the optimisation algorithm.

In the next section, we propose that using optimisation directly on models makes it much easier to define such domain specific operators, substantially reducing the need for repair. As discussed in Sect. 2, we are not the first to propose this: Burton *et al.* [17,4] first proposed the idea and gave an example. They do not, however, provide a general implementation. Denil *et al.* [12] provide a first generic implementation using a transformation scheduling language. However, it is difficult to see how their approach would generalise to population-based algorithms as typically used for multi-objective optimisation as the actual candidate-solution model is implicit in the specification. As a result, their implementation works well for single-state search algorithms, including with backtracking, but cannot easily represent populations of more than one candidate solution. Moreover, they do not provide general mechanisms for encoding optimisation objectives, initial model generation, or model evolution and breeding. Therefore, there still is substantial need for further research in this area.

4 Model-based Optimisation

Three ingredients are required for any search-based algorithm:

1. A representation of individual candidate solutions;
2. A mechanism for generating new candidate solutions from existing candidate solutions (e.g., through mutation or breeding); and
3. A mechanism for evaluating the quality of candidate solutions; that is how well they satisfy each of the optimisation objectives (often called the solution's fitness).

Most search algorithms also require a means of generating an initial population of candidate solutions. Once these ingredients have been defined for a specific problem, we can apply standard search-based algorithms.

As discussed above, we will use models to represent individual candidate solutions. Therefore, the overall search space is defined through a meta-model. An

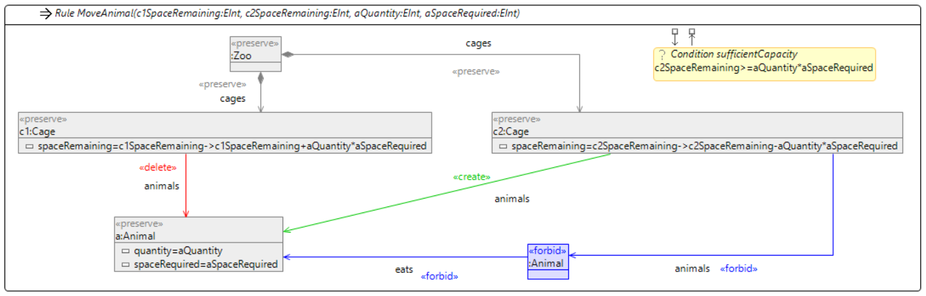

Rule MoveAnimal(c1SpaceRemaining:EInt, c2SpaceRemaining:EInt, aQuantity:EInt, aSpaceRequired:EInt)
«preserve»
:Zoo
cages
«preserve»
Condition sufficientCapacity
c2SpaceRemaining>=aQuantity*aSpaceRequired
«preserve»
cages
«preserve»
c1:Cage
spaceRemaining=c1SpaceRemaining->c1SpaceRemaining+aQuantity*aSpaceRequired
«preserve»
c2:Cage
spaceRemaining=c2SpaceRemaining->c2SpaceRemaining-aQuantity*aSpaceRequired
«delete»
animals
«create»
animals
«preserve»
a:Animal
quantity=aQuantity
spaceRequired=aSpaceRequired
«forbid»
:Animal
eats
«forbid»
animals
«forbid»

Fig. 2. Henshin rule for moving animals between cages

initial population of candidate solutions can be provided in a number of ways—for example, it could be provided as a set of explicit model files or we could use constraint solvers like Alloy [19] to generate a suitable set of initial models (e.g., using the Cartier tool originally developed for transformation testing [20,21]).

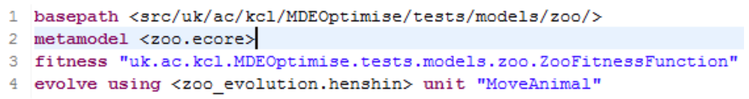
To generate new solutions from existing ones, endogenous model transformations are an obvious candidate. In particular, we propose to use graph transformations, as they have a clear and simple syntax for easily expressing endogenous transformations. For example, Fig. 2 shows a simple Henshin rule that can be used for the search problem described in the previous section. Because these rules are defined on the model level, we will often be able to easily write them in a way that ensures well-formedness rules are preserved.

Evaluating the fitness of candidate solutions can take many different forms. In the simplest case, fitness may be determined by a model query—for example expressed in OCL. In other cases, we may require to run a simulation of the candidate solution, which may involve further transformations etc. (e.g., [2,3]).

To test out these ideas, we are currently developing a prototype tool for model optimisation.[6] Our tool provides a simple Xtext-based DSL to allow describing model-based search problems together with an interpreter for running searches. Search algorithms, fitness functions, and initial model provision are modularised behind Java interfaces. For search algorithms this means that it is easy to incorporate existing implementations, such as the MOEA framework[7]. We currently have no DSL-level support for fitness functions and initial model generation, but plan to add these features. For now, they are specified by providing Java implementations. Solution evolution is realised by Henshin transformations.

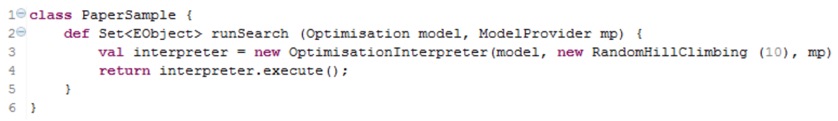
Figure 3 shows an example specification of the `Zoo` example in our tool. After some configuration information in the first line, this code declares the structure of the search space by indicating a meta-model, and then defines relevant fitness functions and model evolvers. Fitness functions are currently provided by implementing a specific Java interface; we are planning to provide full OCL integration

[6] See `https://github.com/mde-optimiser/mde_optimiser`
[7] See `http://moeaframework.org/`



```
basepath <src/uk/ac/kcl/MDEOptimise/tests/models/zoo/>
metamodel <zoo.ecore>
fitness "uk.ac.kcl.MDEOptimise.tests.models.zoo.ZooFitnessFunction"
evolve using <zoo_evolution.henshin> unit "MoveAnimal"
```

Fig. 3. Specification of the `Zoo` search problem



```
class PaperSample {
    def Set<EObject> runSearch (Optimisation model, ModelProvider mp) {
        val interpreter = new OptimisationInterpreter(model, new RandomHillClimbing (10), mp)
        return interpreter.execute();
    }
}
```

Fig. 4. Basic code for running model-based search algorithms

in the language for simple model queries. Evolvers are defined by specifying a Henshin model and naming a rule in this model.

Figure 4 shows how to run a search using our tool. At this point, we only support programmatic invocation. For this, a new `Interpreter` object needs to be created and configured with a parsed version of the problem, a `ModelProvider` for generating initial models, and a generic search algorithm (a variant of random hill climbing in this example). Invoking `execute` runs the search as specified and returns the set of solutions found.

As a tool, our prototype is closest to MOMoT [6], which also uses metamodels to represent search spaces and Henshin rules to represent evolution. However, they are searching for transformation chains using a genetic encoding, while we are searching directly on models.

5 Research Challenges

Our initial work has identified a number of challenges requiring further research to enable model-based optimisation to be used effectively.

5.1 Reuse of existing optimisation algorithms

Some existing optimisation algorithms make particular assumptions about the search space. For example, hill climbing, a basic single-objective search algorithm, expects to be able to identify the complete "neighbourhood" of a given candidate solution so that this can be systematically explored. Similarly, swarm-based search algorithms expect to be able to identify a "direction" vector between solutions and to use this to guide the derivation of one solution from another. These notions are easily defined in classic search-based approaches, where solutions are represented by (high dimensional) numerical vectors. It is less obvious what they mean for models, which are only indirectly related by model transformation chains. Providing appropriate interpretations of these notions will make it possible to reuse more existing search and optimisation algorithms. Beyond that, however, there is an opportunity to explore novel search algorithms that take guidance from the structure and constraints encoding the search space in a model-driven context.

5.2 Model evolution

Generating new candidate solutions from existing ones is a key part of any search algorithm. In model-based search, a number of challenges need to be addressed:

Model breeding. Many population-based algorithms rely on a notion of "breeding", which allows creating new candidate solutions by combining two good parent solutions. This is useful because it allows the search to reach new areas of the search space, hopefully benefiting from the advantages of both parent solutions. For example, in genetic algorithms, "breeding" is realised through so-called crossover operators, which combine two genes by swapping sub-sequences. While mutation of solutions is easily captured by model transformations as discussed, it is less clear how to express breeding. Two approaches seem worth exploring:

1. *Domain-specific breeders.* As with model mutation, we could use model transformations to express domain-specific breeding. These transformations would take in two models and produce a new model. For example, in our `Zoo` problem, we could consider developing a transformation that takes two cage–animal allocations and produces a new one mixing allocations from both sources while making sure that constraints are not violated (e.g., updating `spaceRemaining` values and checking for `eats` relationships). Burton *et al.* [17] show a first example of this for problems where solutions are essentially sets of links between pre-existing model elements.
2. *Generic breeding through model merging.* Model breeding essentially requires identifying the common and different parts of two models so that the common parts can be retained in the new solution and the different parts can be mixed suitably. This is very similar to what has been developed in the context of work on model differencing and model merging [22,23,24,25]. It should be possible to reuse ideas from this field to develop generic model breeders. The key challenge here is that mixing of differences should lead to a new model that is (a) different from both parent models, and (b) well-formed. This will require suitable adjustments to be made to existing diff/merge algorithms for models.

It is very likely that in either case we will not be able to produce breeders that are guaranteed to produce well-formed models, introducing the need to deal with invalid solutions in the search. Abdeen *et al.* [13] give a good discussion of these issues in the context of genetic optimisation of model-transformation chains, where they use repair as well as customised ranking rules. Similar techniques could be applied to model-based optimisation, too.

Efficient model evolution. Our current prototype first establishes all matches for all evolvers and then randomly selects one of them when asked to produce a new candidate solution. Especially where rules have similar pre-conditions or will often not be applicable this seems an inefficient approach. We should explore mechanisms for selecting evolvers (and matches) more efficiently. Denil *et al.* [12] provide some initial insights into this problem by considering optimisation algorithms to be a kind of transformation scheduling specification. This enables them to use different sets of evolvers at different stages of the optimisation process.

Non-deterministic matching in graph transformation engines. Search-based algorithms rely on an amount of randomness underlying the exploration process. Using graph transformations as model evolvers requires the matching process to be non-deterministic. In other words, if there are multiple potential matches for a graph-transformation rule in a model, the choice of match to apply should be non-deterministic. Otherwise, we risk excluding large parts of the search space from the search as a result of an accidental systematic effect of how models happen to be stored in memory or of how model elements are enumerated to find potential matches. Henshin is currently not non-deterministic in this sense, requiring us to explicitly establish all matches and make a random selection. It would be more efficient if Henshin were to non-deterministically select matches on its own.

Compact representations of solutions and evolvers. Typically, a substantial part of a candidate solution will remain constant, as it essentially describes problem constraints rather than solution elements. Burton *et al.* [17] use different models to represent these static parts independently of the parts that change during search. This makes for a very compact solution representation, but requires a separate composition transformation whenever a solution's fitness is to be evaluated or when a new solution needs to be generated. There is a need to understand other similarly compact representations of candidate solutions and how they affect solution evolution and fitness evaluation. Similarly, we should explore ways in which evolvers can be represented more compactly and executed more efficiently knowing that large parts of a candidate solution never change. This has been partially explored in the context of single-state search in the work on design-space exploration [15], where critical-pair analysis is used to establish a dependency graph between evolvers and use this as a basis for guiding the selection of the next evolver to apply.

5.3 Performance

Search algorithms are computationally expensive. Typically, they require a large number of iterations to be run for large populations of candidate solutions. Each iteration requires each candidate solution in the population to be evolved to a new solution and the fitness of these solutions to be evaluated. The performance of search algorithms is therefore substantially influenced by the performance of solution evolution and fitness evaluations. Depending on the size of the models, model transformations can be computationally expensive to execute. There has been recent interest in increasing the efficiency of model transformation execution [26,27]. We need to explore how this could be integrated into model-based optimisation. Ideally, we would like to be able to run model-based search even at system run time to support self-aware system adaptation.

5.4 Flexible definition of model providers

As discussed above, candidate solutions in model-based optimisation are particular in that substantial parts of the model will remain constant as they are describing the search problem. When using constraint solvers like Alloy [19], this would result in a large number of constraints, potentially impacting the performance of initial model generation. Better techniques need to be studied that can limit the performance impact on model generation. It would be useful to identify what are the key constraints that must be available to the constraint solver for it to construct valid instance *of the variable model part* without necessarily recreating the complete constant part every time.

5.5 Expressing fitness evaluations

Some fitness functions are essentially model queries, which can be efficiently expressed in languages like OCL. However, other evaluations will be more complex, including simulations and model analyses. At the moment, these are handled by providing a general Java interface to be implemented for specific fitness evaluations. Techniques better aligned with model-driven approaches need to be developed. Kessentini *et al.* [5] have made some interesting first proposals in this area, which need to be explored further.

6 Conclusions

Search-based software engineering (SBSE) and model-driven engineering (MDE) are highly complementary approaches to software engineering. As a result, there has been substantial interest recently in exploring the combination of both approaches, in particular using MDE technologies to simplify and streamline the application of SBSE. This paper adds to the debate by

1. Providing an overview and initial classification of the current state of the art;
2. Identifying issues with generic genetic encodings of models;
3. Presenting an initial prototype for model-based optimisation; and
4. Identifying a catalogue of research challenges towards complete support for model-based optimisation.

References

1. Harman, M., Jones, B.F.: Search-based software engineering. Information and Software Technology **43**(14) (2001) 833–839

2. Efstathiou, D., McBurney, P., Zschaler, S., Bourcier, J.: Efficient multi-objective optimisation of service compositions in mobile ad hoc networks using lightweight surrogate models. JUCS **20**(8) (2014) 1089–1108 Special issue on WAS4FI 2013.
3. Chatziprimou, K., Lano, K., Zschaler, S.: Surrogate-assisted online optimisation of cloud iaas configurations. In: IEEE 6th Int'l Conf. Cloud Computing Technology and Science (CloudCom). (2014) 138–145
4. Burton, F.R., Poulding, S.: Complementing metaheuristic search with higher abstraction techniques. In: 1st Int'l Workshop Combining Modelling and Search-Based Software Engineering (CMSBSE'13). (2013) 45–48
5. Kessentini, M., Langer, P., Wimmer, M.: Searching models, modeling search: On the synergies of SBSE and MDE. In: 1st Int'l Workshop Combining Modelling and Search-Based Software Engineering (CMSBSE'13). (2013) 51–54
6. Fleck, M., Troya, J., Wimmer, M.: Marrying search-based optimization and model transformation technology. In: Proc. 1st North American Search Based Software Engineering Symposium (NasBASE'15). (2015) Preprint available at `http://martin-fleck.github.io/momot/downloads/NasBASE_MOMoT.pdf`.
7. Drago, M.L., Mirandola, R., Ghezzi, C.: QVTR^2: a rational and performance-aware extension to the relations language. In: Proc. 3rd Int'l Workshop on Non-functional Properties in Domain-Specific Modelling Languages (NFPinDSML'10). (2010)
8. Drago, M.L., Ghezzi, C., Mirandola, R.: Towards quality driven exploration of model transformation spaces. In Whittle, J., Clark, T., Kühne, T., eds.: Proc. 14th Int'l Conf. Model Driven Engineering Languages and Systems (MODELS 2011). Volume 6981 of Lecture Notes in Computer Science., Springer (2011) 2–16
9. Drago, M.L., Ghezzi, C., Mirandola, R.: A quality driven extension to the qvt-relations transformation language. Computer Science – Research and Development **30**(1) (2015) 1–20 First online: 24 November 2011.
10. Williams, J.R.: A Novel Representation for Search-Based Model-Driven Engineering. Phd thesis, University of York (2013)
11. Efstathiou, D., Williams, J.R., Zschaler, S.: Crepe complete: Multi-objective optimisation for your models. In: Proc. 1st Int'l Workshop on Combining Modelling with Search- and Example-Based Approaches (CMSEBA'14). (2014)
12. Denil, J., Jukss, M., Verbrugge, C., Vangheluwe, H.: Search-based model optimization using model transformations. In Amyot, D., Mussbacher, G., eds.: Proc. 8th System Analysis and Modelling Conf. (SAM'14). (2014)
13. Abdeen, H., Varró, D., Sahraoui, H., Nagy, A.S., Debreceni, C., Hegedüs, Á., Horváth, Á.: Multi-objective optimization in rule-based design space exploration. In Crnkovic, I., Chechik, M., Grünbacher, P., eds.: Proc. 29th ACM/IEEE Int'l Conf. Automated Software Engineering (ASE'14), ACM (2014) 289–300
14. Horváth, Á., Varró, D.: CSP(M): Constraint satisfaction problem over models. In Schürr, A., Selic, B., eds.: Proc. Int'l Conf. on Model Driven Engineering Languages and Systems (MoDELS'09). Volume 5795 of LNCS., Springer (2009) 107–121
15. Hegedüs, Á., Horváth, Á., Ráth, I., Varró, D.: A model-driven framework for guided design space exploration. In: Proc 26th IEEE/ACM Int'l Conf. Automated Software Engineering (ASE'11). (November 2011) 173–182
16. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation **6**(2) (April 2002) 182–197
17. Burton, F.R., Paige, R.F., Rose, L.M., Kolovos, D.S., Poulding, S., Smith, S.: Solving acquisition problems using model-driven engineering. In Vallecillo, A., Tolvanen, J.P., Kindler, E., Störrle, H., Kolovos, D., eds.: Proc. 8th European

Conf. on Modelling Foundations and Applications (ECMFA'12). Volume 7349 of LNCS., Springer (2012) 428–443

18. Mandow, L., Montenegro, J.A., Zschaler, S.: Mejora de una representación genética genérica para modelos. In: Actas de la XVII Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA 2016). (2016) in press.
19. Jackson, D.: Alloy: A lightweight object modelling notation. ACM Trans. Softw. Eng. Methodol. **11**(2) (April 2002) 256–290
20. Sen, S., Baudry, B., Mottu, J.M.: On combining multi-formalism knowledge to select models for model transformation testing. In: Proc 1st Int'l Conf. Software Testing, Verification, and Validation. (2008) 328–337
21. Sen, S., Baudry, B., Mottu, J.M.: Automatic model generation strategies for model transformation testing. In Paige, R.F., ed.: Proc. 2nd Int'l Conf. on Theory and Practice of Model Transformations (ICMT'09). Volume 5563 of Lecture Notes in Computer Science., Springer-Verlag (2009) 148–164
22. Kolovos, D.S.: Establishing correspondences between models with the epsilon comparison language. In Paige, R.F., Hartman, A., Rensink, A., eds.: Proc. 5th European Conf. on Model Driven Architecture – Foundations and Applications (ECMDA-FA'09). Volume 5562 of Lecture Notes in Computer Science., Springer (2009) 146–157
23. Kolovos, D.S., Ruscio, D.D., Pierantonio, A., Paige, R.F.: Different models for model matching: An analysis of approaches to support model differencing. In: Proc. ICSE Workshop on Comparison and Versioning of Software Models (CVSM'09), IEEE Computer Society (2009) 1–6
24. Maoz, S., Ringert, J.O., Rumpe, B.: A manifesto for semantic model differencing. In Dingel, J., Solberg, A., eds.: Models in Software Engineering: Workshops and Symposia at MODELS 2010, Reports and Revised Selected Papers. Springer Berlin Heidelberg (2011) 194–203
25. Langer, P., Mayerhofer, T., Kappel, G.: Semantic model differencing utilizing behavioral semantics specifications. In Dingel, J., Schulte, W., Ramos, I., Abrahão, S., Insfran, E., eds.: Proc. 17th Int'l Conf. Model-Driven Engineering Languages and Systems (MODELS'14), Springer International Publishing (2014) 116–132
26. Amstel, M., Bosems, S., Kurtev, I., Pires, L.F.: Performance in model transformations: Experiments with ATL and QVT. In Cabot, J., Visser, E., eds.: Proc. 4th Int'l Conf. Theory and Practice of Model Transformations (ICMT'11), Springer Berlin Heidelberg (2011) 198–212
27. Mészáros, T., Mezei, G., Levendovszky, T., Asztalos, M.: Manual and automated performance optimization of model transformation systems. International Journal on Software Tools for Technology Transfer **12**(3) (2010) 231–243