Crepe Complète: Multi-objective optimisation for your models*

Dionysios Efstathiou[1], James R. Williams[2], and Steffen Zschaler[1]

[1] Department of Informatics, King's College London
dionysios.efstathiou@kcl.ac.uk, szschaler@acm.org
[2] Department of Computer Science, University of York
james.r.williams@york.ac.uk

Abstract. Search-based software engineering views software development as a process of searching through the design space for an optimal solution according to some quality criteria. It seems natural to try and build automated implementations of this idea based on concepts from model-driven engineering—using meta-models as characterisations of design spaces and model transformations as algorithms / heuristics for the exploration of the design space. Yet, there is only very limited research in this area. In this paper, we contribute to the discussion in two ways: 1) we provide an extension of Crepe, a framework for single-objective optimisation based on the Epsilon tool set, to support multi-objective optimisation, and 2) we present an experimental comparison between a multi-objective optimisation problem implemented in the extended Crepe and the same problem implemented in native Java using an optimised internal representation of the search space and of solution candidates. The experiment highlights key areas of improvement required in Crepe to enable it to fully compete with bespoke implementations.

1 Introduction

Search-Based Software Engineering (SBSE) [1] views software development as a process of searching a (potentially large) design space for a design optimally addressing the given requirements. The goal, then, is to reduce the need for humans to perform this search, which would be slow and potentially unreliable or error prone. Instead, SBSE research aims to identify ways of automating the search, using techniques from meta-heuristic search or operations research. The field has grown over the past decade; there are a number of fields in software development where automated search can be applied with good results [1].

Model-driven engineering (MDE) [2] aims to develop software systems using models; that is, at a higher level of abstraction than offered by currently prevalent programming languages. Because of this higher level of abstraction, often combined with the use of domain-specific modelling languages, a number of processes in software development can be automated or semi-automated. Centrally,

* The work reported in this paper has been partially funded by the European Union under FP7 ITN RELATE and FP7 STREP OSSMETER.

MDE propagates the use of model transformations [3] to automate analysis of models or generation of fully executable implementations.

It seems obvious that a combination of MDE and SBSE could benefit from the advantages of each approach: SBSE studies specific techniques for searching specific kinds of design spaces, but lacks a general technique for simply and systematically implementing such searches for different design spaces. MDE, on the other hand, provides good techniques for automating processes in software development in a generalisable way, but lacks specific support for search- or optimisation-based techniques. A combination of the two approaches could provide a framework of model-transformation techniques to enable effective implementation of search algorithms for specific domains.

Two problems need to be solved to combine MDE and SBSE:

1. We need to define general implementations of search algorithms and identify the parameters required to specialise these for particular domains. At a minimum, these parameters will need to be in the form of meta-models that allow the description of the domain (i.e., of the design space to be searched).
2. We need to ensure search is efficient. Search-based algorithms can take a long time to run, so performance considerations are important in their design.

Note that the two issues are potentially in conflict: realising a generic implementation may induce an additional performance hit. Performance in optimisation algorithms is substantially influenced by the representation of candidate solutions and the efficiency with which these can be manipulated. A generic, model-based implementation may cause problems in this regard.

In this paper, we address the two above problems in the following ways:

1. We present a set of meta-models and generic model-driven framework implementing meta-heuristic, multi-objective optimisation. This is based on Crepe [4], which supports single-objective optimisation already.
2. We compare the performance of this framework with the performance of a hand-crafted implementation for a specific SBSE problem and provide a first discussion of the trade off between ease of implementation (aided by the generality of the framework) and loss of efficiency.

2 Related Work

While there is some work on applying search-based or optimisation-based techniques with MDE (see [4, 5] for some examples), work on generic support for search-based or optimisation-based exploration of models is still rare. To the best of our knowledge, only two other works exist:

QVTR^2 [6] propose an extension of QVTR where different alternatives are encoded as non-deterministic relations. These are incrementally rewritten into deterministic relations by 1) making a non-deterministic choice, 2) asking a developer to compare the outcome of that choice to results produced by another

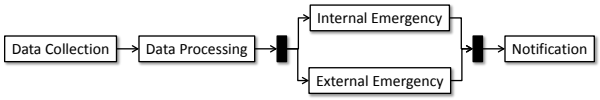

Fig. 1. The workflow of the example composite application

choice, and 3) fixing the preferred choice by rewriting the original transformation specification. While this enables the implementation of optimisation-based approaches, it requires a substantial amount of manual intervention.

The approach recently proposed by Denil *et al.* [7] also encodes the design space partially as transformation rules. Search strategies are encoded in the rule-scheduling, which can be freely defined in the underlying transformation system FTG+PM. The design space is further constrained by providing a fixed set of model elements that can be recombined by the transformation rules. Beyond this, however, the proposed approach seems to require a complete reimplementation of both the rules and the search strategy for every new problem.

The use of meta-model–like structures to represent candidate solutions in evolutionary optimisation has first been explored by Christopher Simons in [8]. Canfora *et al.* [9] were the first to use genetic algorithms for discovering QoS-optimal service compositions. An extensive survey of various optimisation techniques applied to the service composition search problem can be found in [10].

3 Motivating Example

Consider a system for improving the decision making of firefighters in emergency situations. Firefighters are equipped with mobile devices with various networking capabilities. These devices form an infrastructure-less mobile *ad hoc* network enabling them to offer their resources—such as application data and network—as software services. Service composition enables devices to combine multiple services into composite applications in order to satisfy complex functional goals.

For example, consider a forest fire where a commanding firefighter uses a composite application to identify endangered firefighters and react appropriately. The commander aggregates information about the condition of his subordinates (position, heart rate, oxygen level). This data needs to be fed to a processing service which assesses if something is going wrong. In the case of an emergency event (e.g., a firefighter has stopped moving and high levels of carbon dioxide in the blood are observed), another team in the close proximity must be notified to intervene along with an external medical team which must be ready to approach. Figure 1 depicts the underlying composite application as a work flow.

Availability of resources (e.g., network bandwidth, battery power) strongly affects the quality of the services offered. Different choices of concrete services (and hosting nodes) for each of the abstract services in Fig. 1 lead to different overall quality of service (QoS). Finding service compositions with optimal QoS

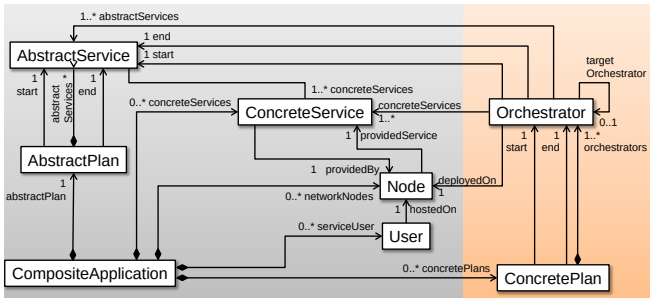

Fig. 2. The metamodel for defining distributed service composition configurations

trade-offs is challenging. The size of the search space makes meta-heuristic search techniques [11] interesting candidates for addressing this problem [12].

Figure 2 describes the search space of the firefighter problem using a meta-model. Each instance of the meta-model represents a candidate solution, whose QoS can be analysed—for example, using a simulation. The parts of the meta-model shaded in grey represent the problem description; that is, the abstract work flow as well as the available nodes and concrete services. Any candidate solution for the same problem will share the same instances of this part. In contrast, the elements shaded in orange represent a specific candidate solution by specifying the specific concrete services and orchestrators selected.

4 Crepe – Search-Based Algorithms for MDE

Meta-heuristic search-based optimisation algorithms are commonly implemented with respect to an *encoding* of the problem domain (traditionally binary or integer vectors). This encoding (the *genotype*) is translated into the native format (the *phenotype*) of the solution to be evaluated by a fitness function. *Genetic operators* (mutation and crossover) act upon the encoded form of solutions, modifying and combining different solutions, with the intention of producing new solutions with higher fitness. *Crepe* [4] provides such an encoding, and associated genetic operators, for MDE models. Crepe can encode the space of models that conform to any given meta-model, enabling metaheuristic optimisation techniques to be applied to any problem whose solution can be represented by a meta-model instance. Crepe is implemented in the *Epsilon Object Language* (EOL) [13] and processes models based on the Ecore meta-modelling language [14]. In this section, we briefly overview the key features and limitations of Crepe. A full description of Crepe can be found in [4].

4.1 Crepe Components

There are three main components of Crepe: the structure of the genotype, the finitisation information, and the mappings between genotype and phenotype. In

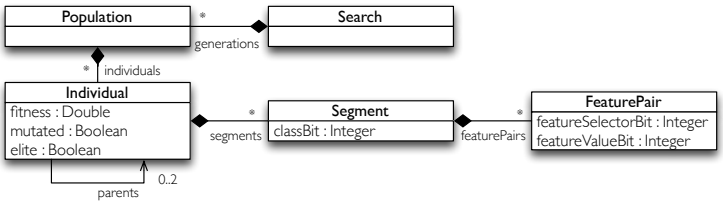

Fig. 3. The metamodel that defines the structure of individuals in Crepe

the spirit of MDE, all components in Crepe are either models or model transformations. Fig. 3 shows the meta-model that describes a single execution of a search algorithm. Genotypic *individuals* are grouped into *populations*. The genotype of candidate solutions is a structured sequence of integers: individuals are composed of *segments*, which are composed of *feature pairs*. Each segment represents a single object in the model being encoded. Feature pairs are used to assign values to the different features (attributes, references) of that object. Generic model transformations translate individuals into models that conform to a user-provided meta-model (and *vice versa*).

To make these transformations possible, the user needs to define a *finitisation model* – a model containing information that constrains the model space. For instance, when generating model objects that have string-valued attributes, the finitisation model can be used to restrict the set of strings that can be assigned to objects with that attribute. Finitisation models can also restrict structures as well as data; for instance, enforcing an upper bound on a reference, or stating that certain meta-classes should not be instantiated.

Each transformed individual is passed to a user-defined fitness function for evaluation. This function returns a double which Crepe assigns to the individual's fitness attribute, for search algorithms to consume.

4.2 Limitations: Single Objective Only and Meta-Model Constraints

To date, Crepe has only supported problems that have a single fitness value. Many real-world problems, however, are multi-objective and currently these different objectives would need to be aggregated to be used in Crepe. Moreover, meta-models are only able to define certain structural characteristics of conforming models; in order to define more complex constraints, such as class invariants or relationships between features, languages such as the Object Constraint Language [15] are used. Crepe does not support these extra constraints, meaning that the model space encoded can contain models that are structurally correct, but semantically invalid, thereby requiring an additional filtering step.

4.3 Extending Crepe

We have extended Crepe to address these limitations:

Fig. 4. The multi-objective specialisation of the genotype metamodel

Support for Multi-Objective Optimisation In order to support multi-objective optimisation algorithms in Crepe, we have specialised the search metamodel. As shown in Fig. 3, individuals only have a single fitness attribute. We therefore specialise the Individual class to enable individuals to have multiple fitness values, as shown in Fig. 4. Objectives are represented by their own metaclass and have both a fitness score and a flag that specifies whether the objective should be minimised or maximised.

Whereas in the single objective version of Crepe, the user-defined fitness function returns a double representing the fitness, for multi-objective problems it needs to return a sequence of Objective objects—one for each of the problem's objectives. Crepe assigns the collection to the individual's objectives reference (Fig. 4), and multi-objective optimisation algorithms can use these, and Crepe's existing genetic operators [4], to guide the search.

Support for Model Constraints To support constraints, and therefore ensure Crepe produces semantically valid models, we provide a means for the user to programmatically define finitisations for specific features. Whilst transforming a segment into a model object, Crepe now delegates the job of selecting appropriate values for that object's features to a user-defined function. This means that the user can select the appropriate finitisation values based on the current state of the object. If the user-defined function returns a *null* value for that feature, Crepe will compute the feature's standard set of finitisations. Otherwise, the values returned by the user-defined function are used.

For example, a constraint in the fire-fighting case study is that orchestrators must be deployed on separate nodes (the deployedOn reference). By default, Crepe will indiscriminately select nodes to be assigned to that reference. Using the new finitisation-constraint method, the user can specify to only select nodes that haven't previously been assigned to an orchestrator.

5 Experimental Evaluation

To evaluate how Crepe compares to a problem-specific optimisation tool, we apply it to the fire-fighting scenario. Both Crepe and the bespoke technique presented in [12] use the NSGA-II [16] optimisation algorithm, but differ in their implementation: Crepe uses the generic methods presented in the preceding sections, while [12] uses a bespoke implementation and encoding.

In this section, we first briefly discuss the Crepe implementation of the fire-fighting scenario, before giving some more details on our experimental setup and reporting the results of our experiment.

Fig. 5. The modification made to the service composition metamodel from Fig. 2. Note that the remaining classes and feature of Fig. 2 remain the same

5.1 Implementing the Firefighter Scenario in Crepe

The solution to the scenario is a model that conforms to the meta-model in Fig. 2. Crepe does not need to find the entire model, however, as the set of abstract and concrete services is predetermined. The goal is to discover the optimal configuration of nodes that orchestrate appropriate services. An input to Crepe therefore is an existing service composition model containing the available services. For each solution, Crepe creates a copy of this model with the appropriate orchestrators. In the finitisation model we specify that we don't want Crepe to produce abstract or concrete services, only allowing Crepe to create Orchestrator objects. We also use the input model to finitise the solutions' orchestrators with the predefined set of abstract and concrete services.

Ensuring the constraints of the service-composition model required the introduction of a new meta-class. In its existing form (Fig. 2), there is no explicit relation between the abstract and concrete services provided by an Orchestrator meaning that arbitrary selections of services are structurally valid but semantically incorrect: for each orchestrated abstract service, one of its associated concrete services must also be orchestrated. To address this, we have introduced a new meta-class tying together abstract and concrete services, as shown in Fig. 5. The user-defined finitisation operation (cf. Sect. 4.3) finitises the cService reference only with valid concrete services with respect to the selected abstract service. Implementing the context-specific finitisation operation took 38 lines of EOL and covered constraints for six different meta-model features. The full set of constraints and code to run the case study is available on-line.[3]

The bespoke solution [12] uses surrogate functions to approximate fitness of candidate solutions and make the optimisation feasible. We first collected a dataset of solutions evaluated on a detailed but computationally expensive simulation tool. Based on this dataset, we built surrogate functions using the lightweight multivariate adaptive regression splines technique [17]. The Crepe-based implementation reuses the developed surrogate functions. Specifically, we considered response time, energy consumption, and service availability for a candidate service composition.

[3] https://github.com/efstathiou/crepe/tree/dev/crepe.examples/service-composition-de

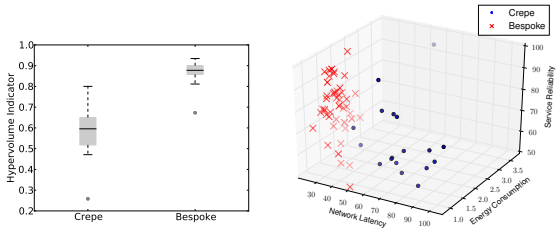

Fig. 6. Hypervolume indicator results (left) and 3D Pareto fronts (rights) achieved by the two implementations

5.2 Experimental Setup

We created a network of 84 nodes (4 group, 16 engine, and 64 team members and leaders) which offer the software services mentioned in Sect. 3 and cooperatively form an infrastructure-less mobile *ad hoc* network. The initial state of the network was determined through a simulation run and was then made available to both implementations. We ran both techniques for 30 generations with a population size of 96. For statistical significance each run was repeated 30 times and used the Mann-Whitney test [18]. We used Cohen's d [18] to estimate the effect size of the difference between the results achieved.

5.3 Results

We were interested in comparing both the performance and the quality of the optimisation in both implementations.

First, we compared the execution time of the two implementations. A single run of the NSGA-II for 30 generations lasts on average $\sim$ 43 minutes for Crepe and $\sim$ 1 minute for the bespoke implementation using a machine with Intel Core i7 vPro with 12GB DDR3 RAM, running the Linux 3.2.0-40 kernel.

Next, we compared the quality of the results produced after 30 generations. Figure 6 shows the hypervolume indicator achieved by the NSGA-II algorithm for both the Crepe and the bespoke Java-based implementation. The Mann-Whitney test results show that the results achieved by the bespoke implementation were significantly better ($p - value = 2.4876 \cdot 10^{-11}$) than those of Crepe with a large Cohen effect size.

6 Discussion

Our experiment shows that, while Crepe produces solutions of acceptable quality, they are less good than the solutions produced by a bespoke implementation and it takes substantially longer for Crepe to compute them. In this section, we discuss these results in some more detail.

We expected Crepe to be somewhat slower than the bespoke implementation because of a) its generic nature and b) its use of models (i.e., explicit object structures) rather than vectors to represent the genotype of candidate solutions. We

have been somewhat surprised by the substantial performance hit incurred and are currently investigating the reasons in detail. However, two things should be kept in mind when looking at Crepe's performance: 1) the time taken would still be acceptable in the context of a wider software-development activity (though not for on-line use) and 2) the optimisation performance should be weighed against the time saved in building the optimisation algorithm. One potential solution to the performance problem is to replace the slowest parts of Crepe with highly optimised Java-based implementations which can be invoked from the interpreted EOL.

Even after tuning its parameters, Crepe favoured solutions with fewer orchestrators and therefore the discovered Pareto front was on average less optimal than the bespoke implementation's front (Fig. 6). In our initial experiments, we found that Crepe was biased towards solutions with between one and three orchestrators, essentially optimising more centralised solutions and producing Pareto fronts with a low hypervolume. We have previously shown [12] that centralised orchestrations are not optimal for distributed networks. To address this, we tuned the parameters that determine the sizes of individuals produced by Crepe to optimise for a size that produces solutions without such a strong bias. Crepe does not guarantee that all features are assigned values. For instance, if there are fewer feature pairs in a segment than features in the encoded model object, some features will be left empty. We configured the number of segments and features to allow space for all data required. Even so, we found that there were occasional solutions ($< 5\%$) with missing features that made them invalid. This could be down to the custom finitisations forcing the search algorithm to be too restrictive. To address this, the fitness function deterministically selects appropriate values for any missing features. However, this may introduce bias as it essentially ignores the heuristics and fitness evaluations for these features. Note, however, that the results indicate (Fig. 6) that, in some iterations, Crepe does explore near-optimal solutions with various orchestrations and achieves results similar to the bespoke approach.

7 Conclusions and Outlook

This paper has explored the challenge of general-purpose model-based optimisation frameworks, and presented Crepe's first foray into the multi-objective optimisation domain. We have described the required extensions to Crepe: support for multiple objectives, and a new approach to finitisation which allows users to have more control over the encoded models. By comparing Crepe against a highly optimised problem-specific representation, we found that Crepe was able to discover equally optimal solutions, but on average performed worse on this specific problem. As expected, the execution time of Crepe was much longer, however still within a range that would be acceptable during development. We believe that a generic model-based framework like Crepe can be a good starting point for combining MDE and SBSE. However, research is still needed to improve the quality of the solutions as well as the performance of the optimisation.

In particular, we plan to extend the scalability analysis of Crepe performed in [4] by applying it to real-world problems defined using different size metamodels.

References

1. Harman, M., McMinn, P., Souza, J.T., Yoo, S.: Search Based Software Engineering: Techniques, Taxonomy, Tutorial. In: Empirical Software Engineering and Verification. Volume 7007. (2012) 1–59
2. Schmidt, D.C.: Model-Driven Engineering. IEEE Computer **39**(2) (February 2006) 25–31
3. Sendall, S., Kozaczynski, W.: Model Transformation: The Heart and Soul of Model-Driven Software Development. IEEE Software **20**(5) (September 2003) 42–45
4. Williams, J.R.: A Novel Representation for Search-Based Model-Driven Engineering. PhD thesis, University of York (2013)
5. 1st Int'l Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE'13). In: 1st Int'l Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE'13). (2013)
6. 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)
7. 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)
8. Simons, C.L.: Interactive, Evolutionary Computing in Early Lifecycle Software Engineering Design. PhD thesis, University of the West of England (May 2011)
9. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: An Approach for QoS-Aware Service Composition Based on Genetic Algorithms. In: GECCO. (2005)
10. Strunk, A.: QoS-Aware Service Composition: A Survey. In: Proc. 8th European Conference on Web Services. (2010)
11. Blum, C., Roli, A.: Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. ACM Comput. Surv. **35**(3) (2003) 268–308
12. Efstathiou, D., McBurney, P., Zschaler, S., Bourcier, J.: Surrogate-Assisted Optimisation of Composite Applications in Mobile Ad hoc Networks. In: Proc. ACM Genetic and Evolutionary Computation Conference (GECCO'14). (2014)
13. Kolovos, D.S., Paige, R.F., Polack, F.A.C.: The Epsilon Object Language (EOL). In: Model Driven Architecture — Foundations and Applications. Volume 4066 of Lecture Notes in Computer Science., Springer (2006) 128–142
14. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF Eclipse Modeling Framework. Second edn. The Eclipse Series. Addison-Wesley (2009)
15. Object Management Group: UML 2.0 OCL Final Adopted Specification. OMG document ptc/2003-10-14 (October 2003) URL http://www.omg.org/cgi-bin/doc?ptc/03-10-14.
16. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II. IEEE Trans. Evol. Comp. **6** (2000) 182–197
17. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer (2003)
18. Arcuri, A., Briand, L.C.: A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering. In: ICSE. (2011)