Modelling and Multi-Objective Optimization of Quality Attributes in Variability-Rich Software

Rafael Olaechea, Steven Stewart, Krzysztof Czarnecki, Derek Rayside
University of Waterloo
Wateroo, Ontario
{rolaechea, kczarnec}@gsd.uwaterloo.ca, {steven.stewart, drayside}@uwaterloo.ca

ABSTRACT

Variability-rich software, such as software product lines, offers optional and alternative features to accommodate varying needs of users. Designers of variability-rich software face the challenge of reasoning about the impact of selecting such features on the quality attributes of the resulting software variant. Attributed feature models have been proposed to model such features and their impact on quality attributes, but existing variability modelling languages and tools have limited or no support for such models and the complex multi-objective optimization problem that arises. This paper presents ClaferMoo, a language and tool that addresses these shortcomings. ClaferMoo uses type inheritance to modularize the attribution of features in feature models and allows specifying multiple optimization goals. We evaluate an implementation of the language on a set of attributed feature models from the literature, showing that the optimization infrastructure can handle small-scale feature models with about a dozen features within seconds.

Categories and Subject Descriptors

D.2.13 [**Software Engineering**]: Reusable Software—*Reuse Models*

General Terms

Software Product Lines, Multi-objective optimization

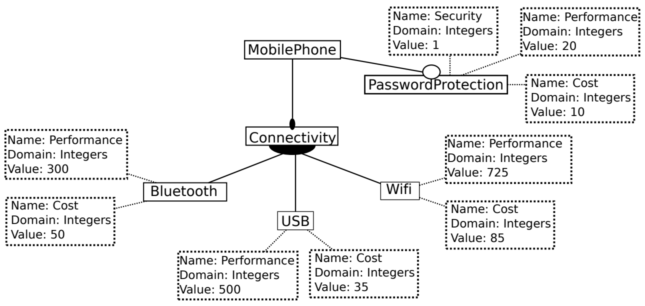
1. INTRODUCTION

Software and variability are ubiquitous. Devices, such as watches and automobiles, come in different variants, and so does the software embedded within them. Companies using software product line engineering derive such variants from a set of configurable software assets [4]. Ecosystem platforms, such as the Linux kernel, offer thousands of configuration options in order to support a wide range of applications [3]. Dynamically adaptable software provides numerous variation points to allow reconfiguration and extension [9]. Variability also occurs when multiple design choices are allowed early in the design of a system.

Variability gives rise to complex decision making that needs to take quality (non-functional) requirements into account. Software designers must determine the range of *features*—coherent increments of functionality—that the software will offer to their customers. For example, a mobile phone platform can offer different types of displays, input devices, communication protocols, and encryption technologies. Each of these features will impact differently the quality attributes of the resulting product, such as battery life and performance. The users of the platform select features for a specific product, while also paying attention to the resulting quality attributes. In particular, they want to be able to select the features required in their application context and automatically determine the selection of other features to *optimize* desired quality attributes of the resulting product. For example, a user might require a particular display type, but then optimize the remaining choices to minimize cost and maximize battery life.

An emerging trend from software product line engineering is to represent software variability explicitly as variability models [9]. Such models enable software designers to decide and manage the scope of the provided variability; they also enable users to select among the available choices and derive a software variant that meets their needs. While the majority of the existing variability modelling approaches and tools do not account for quality attributes, *attributed feature models* address this need [2].

A *feature model* is a menu-like hierarchy of features with additional constraints on valid feature selections [7]. Figure 1 shows a simplified feature model of a mobile phone platform, which can be used to derive a software variant. *Connectivity* is a mandatory feature (denoted by the filled circle), with an OR group of three child features (denoted by the filled arc); that is, at least one of Bluetooth, USB, or Wifi has to be selected for a concrete variant. *PasswordProtection* is an optional feature (denoted by the empty circle). A feature model can also contain cross-tree constraints as logical formulas, such as *Wifi implies PasswordProtection*. An attributed feature model puts "price tags" on features, denoting the feature's impact on some quality attribute of the resulting variant. For example, the added dollar cost of selecting Bluetooth is \$50 versus \$85 for Wifi, and the added performance gain is 300 for Bluetooth versus 725 for Wifi. The *Cost* and *Performance* functions indicate how quality attributes of a variant are computed based on the feature-level quality attributes (feature attributes, for short).



$$Cost_{Product} = \sum_{f \in Product} Cost_f \qquad (1)$$

$$Performance_{Product} = \sum_{f \in Product} Performance_f \qquad (2)$$

Figure 1: An attributed feature model of a MobilePhone platform (adapted from Benavides et al. [2])

While attributed feature models provide an intuitive way to represent the available features and their variability and impact on the quality attributes of the resulting variant, current language and tool support for such models is limited. First, existing work assumes that features are annotated with attribute values on a per need basis, without providing any way to group features into abstractions, nor to type features using these abstractions; consequently, one could easily overlook annotating some features when new attributes are introduced. For example, some features could be purely software while others could be a combination of software and hardware. Certain attributes may apply to certain types of features only. Second, while the simple summation of price tags works well for some quality attributes, software features often interact [14], meaning that the combined contribution of two or more features is not just the sum of their individual contributions. For example, the memory consumption of two features may be higher (or lower) than the sum of the memory consumption of each feature when used in separation. A language for representing attributed feature models has to be expressive enough to accommodate such interactions. Third, the language and the tooling needs to support multiple optimization goals. While existing work has addressed optimization for a single objective or a weighted combination of objectives (*e.g.*, [2]), current variability modelling languages and tools do not support truly multi-objective optimization. Such a capability is highly desirable because collapsing the problem to a single objective fails to explore much of the design space.

This paper addresses these shortcomings by showing how a variability modelling language with support for types and inheritance can express the attribution of feature models in a modular way and also accommodate feature interactions. Further, the paper shows how multiple goals can be concisely defined based on types. We demonstrate these ideas using ClaferMoo, which extends the structural modelling language Clafer [1] with facilities to express multiple optimization objectives. ClaferMoo uses existing extensions of the relational modelling language Alloy [8, 10, 16] and its tooling, which enable it to perform exact, discrete multi-objective optimization in the presence of relational constraints. We evaluate ClaferMoo on a set of existing attributed feature models, showing that they can be adequately expressed in ClaferMoo and that the existing optimization infrastructure can handle small-scale feature models with about a dozen features within seconds. Although small-scale, these feature models represent real-world software and already give rise to thousands or millions of configurations, making tool support necessary. In future work, we plan to improve the scalability of the optimization infrastructure and to test the approach and tooling on a wider set of models.

Paper organization. Section 2 describes how attributed feature models are expressed in Clafer, followed by Section 3 on multi-objective optimization in Clafer. Section 4 presents the toolchain for the product derivation process. Section 5 presents the methodology employed and the experimental results, and Section 6 presents threats to validity. Lastly, Section 7 presents related research, and Section 8 concludes.

2. ATTRIBUTED FEATURE MODELS

Clafer has been designed to provide concise syntax for specifying feature models. Figure 2 shows the basic feature model (without attributes) of the MobilePhone platform from Figure 1 in Clafer. Each line in the Clafer model—except those in brackets—defines a type; the indentation denotes type nesting, exemplified in the corresponding feature model in Figure 1. The question mark following PasswordProtection denotes optionality. More precisely, the question mark is syntactic sugar for the cardinality constraint 0..1; *i.e.*, an instance of MobilePhone can contain zero or one instances of PasswordProtection. Since Connectivity does not have a cardinality constraint at the end of the line, it is assumed to be mandatory (cardinality 1..1). The or keyword in front of Connectivity represents a group cardinality constant, which constrains the possible number of children. More precisely, or stands for 1..*; that is, an instance of Connectivity must contain an instance of any non-empty subset of Bluetooth or Wifi or USB. Note that MobilePhone is declared as abstract, meaning that there are no instances of this type; however, the last line declares MyPhone as a singleton concrete subtype of MobilePhone, effectively defining an instance of the MobilePhone product line. The lines in brackets represent constraints (the lines are conjoined implicitly), and since the entire constraint is nested under MyPhone, it constrains the instance represented by MyPhone. Effectively, MyPhone is a particular instance of MobilePhone that has Bluetooth and USB, but not Wifi or PasswordProtection.

Clafer also supports any attribute that can be encoded as an integer. Figure 3 shows our MobilePhone feature model with performance attributes added, including their values.

When different quality attributes need to be used, it is useful to group them into abstractions from which individual features can inherit. Figure 4 illustrates this idea (ignore the last three lines for now). The first part of the model defines an inheritance hierarchy of features: all features have performance and cost attributes. Security features additionally have a security attribute.

```
abstract MobilePhone
  or Connectivity
    Bluetooth
    USB
    Wifi
  PasswordProtection ?

MyPhone : MobilePhone
  [  no Wifi
    USB
    Bluetooth
    no PasswordProtection ]
```

Figure 2: Feature model of the MobilePhone platform and a sample instance in Clafer

```
abstract MobilePhone
  or Connectivity
    Bluetooth
      performance : integer = 300
    USB
      performance : integer = 500
    Wifi
      performance : integer = 725
  PasswordProtection ?
    performance : integer = 20
```

Figure 3: Attributed feature model in Clafer

MobilePhone in Figure 4 also defines three additional attributes, each starting with total_. These attributes nest constraints that set them up as the total sums of, respectively, the performance, security, or cost attributes of the selected features. The navigation expression Feature.performance returns a set of integers, which are all the performance values nested under all instances of Feature. The sum operator computes the sum of these numbers.

Feature interactions can be easily accommodated by adding conditional terms to the sum. For example, let's say that Bluetooth and Wifi interact negatively in terms of performance, with the negative impact of 20. Such interaction could be expressed as follows:

```
(sum Feature.performance) + (Bluetooth && Wifi ? −20 :0)
```

3. MULTI-OBJECTIVE OPTIMIZATION

A multi-objective optimization problem arises when modelling goals (or objectives) over quality attributes. This occurs when the user intends to maximize or minimize functions over multiple product quality attributes.

A multi-objective optimization problem has a set of solutions known as the *Pareto Front* that represents the tradeoffs between the different objectives [15]. The Pareto Front consists of Pareto-optimal solutions, which can be defined intuitively as follows: a Pareto-optimal solution is one in which no metric can be made better off without making another worse off. As the old adage goes, 'fast, cheap, or good: pick two'. In other words, the Pareto Front of this adage will include three solutions: fast and cheap, fast and good, and cheap and good.

```
abstract Feature
  performance : integer
  cost : integer
abstract SecurityFeature : Feature
  security : integer

abstract MobilePhone
  or Connectivity
    Bluetooth : Feature
      [ performance = 300
        cost = 50 ]
    USB : Feature
      [ performance = 500
        cost = 35 ]
    Wifi : Feature
      [ performance = 725
        cost = 85]
  PasswordProtection : SecurityFeature ?
      [ security = 1
        performance = 20
        cost = 10]
  total_performance : integer
    [ total_performance = sum Feature.performance ]
  total_cost : integer
    [ total_cost = sum Feature.cost ]
  total_security : integer
    [ total_security = sum SecurityFeature.security ]

MyPhone : MobilePhone

<< max MyPhone.total_performance >>
<< min MyPhone.total_cost >>
// << max MyPhone.total_security >>
```

Figure 4: Feature model with inherited attributes in Clafer and with goals (ClaferMoo extension)

Multi-objective solvers may be *heuristic* or *exact*. Heuristic solvers are often built with genetic algorithms. We use an exact solver called Moolloy [10] that was built on top of Kodkod [17] and incorporated into the surface syntax of Alloy in previous work [16].

The underlying ClaferMoo implementation, based on Moolloy, will generate multiple model instances in the presence of multiple optimization goals. For example, the model in Figure 4 specifies two goals—one to maximize performance and another to minimize cost, with goals being expressed within opening and closing angled brackets, followed by "max" or "min" and then an integer expression. (A third goal—maximize security—is commented out because it is easier to visualize a Pareto Front in two dimensions.) Figure 5 shows the Pareto Front for the performance and cost objectives of the mobile phone example.

If needed, MyPhone can be constrained with some required choices, such as the selection of USB, but not Wifi; such constrained instances can be referred to as *partial configurations*. Given a partial configuration, the underlying implementation searches for optimal solutions to complete it.

4. IMPLEMENTATION

Figure 6 illustrates the high-level architecture of the current ClaferMoo implementation. The architecture is a batch-sequential process, where each stack of boxes represents a

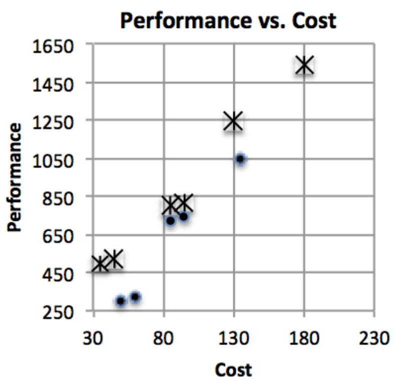


Figure 5: Performance vs. Cost for the mobile phone platform in Figure 4. The Pareto Front, indicated with asterisks, identifies optimal configurations. Dots represent dominated (sub-optimal) solutions that were discovered during the optimization process.

computational node and each arrow represents data flow. The top of each stack represents an existing tool that was reused and extended in the implementation of ClaferMoo; the boxes below represent extensions.

The existing Clafer-to-Alloy translator [1] has been extended with the sum operator, support for expressing goals, and a modified translation to Alloy that enables specifying partial instances, as explained next.

The solving and analysis of Clafer models is implemented by translation to a modified version of Alloy4—a lightweight modelling language described in [5]—using a version of the Alloy Analyzer that has been extended to support multi-objective optimization [10, 16] and partial instances [8]. Until recently, Alloy4 had lacked support for specifying partially completed instances; however, a recent paper [8] introduced new syntax—the *inst* block—that more fully utilizes the capabilities of Kodkod [17]—Alloy's underlying relational reasoner. The Clafer translator produces an Alloy model that includes an *inst* block for a partially configured, concrete attributed feature model, and that also uses the *objectives* block for expressing goals. Thus, both the partial instance and objectives extensions are required.

Ordinarily, the Alloy Analyzer simply generates model instances that are then translated back to Clafer as concrete instances; however, in this toolchain, Moolloy [10]—an exact, discrete multi-objective optimization solver—is introduced into the backend in order to generate model instances that respect the desired optimization goals. Moolloy's underlying algorithm incrementally explores the boundaries of the Pareto Front, making multiple calls to Kodkod, which, in turn, relies on a backend, off-the-shelf SAT-solver.

5. EVALUATION

We have evaluated ClaferMoo on an existing collection of attributed feature models from the literature [14] [12] [13]. The objective of the evaluation was to address two questions:

1. Can real-world attributed feature models be expressed in ClaferMoo?
2. How scalable is the current implementation?

Table 1 lists the models used in the evaluation. These models were extracted from existing, highly configurable open-source software systems—the first column indicates the names of these systems. The configuration options are typically compilation options controlling conditional code inclusion. As indicated in the second column, seven of these models have around a dozen features (*i.e.*, configuration options) each; two have around one hundred features each. The third column indicates the number of cross-tree constraints, which are binary *requires* and *excludes* constraints. The last column indicates the quality attributes included in the models. No feature interactions were considered in our extracted models with respect to quality attributes. The majority of the models include the binary footprint of the executable; one model includes three quality attributes.

5.1 Methodology

Question 1, *expressiveness*, was answered affirmatively in two ways. First, by writing a script to translate attributed feature models from Apel's XML format to ClaferMoo. Second, feature models presented graphically by Siegmund et al. [13] were translated to ClaferMoo by hand. All of the models can be expressed naturally in ClaferMoo.

Question 2, *performance & scalability*, was evaluated by generating and solving ten random, satisfiable, partial configurations for each feature model. These partial configurations were generated by assigning, with equal probability, each feature in a model a status of 'selected', 'not selected', or 'unconstrained.' Partial configurations were checked to ensure that they did not violate any constraints of the feature model; unsatisfiable partial configurations were discarded. Each optimization run was repeated 20 times and median running times were recorded. The optimization goals were to minimize footprint, maximize performance, minimize price, and maximize reliability. The experiments were run on a Macbook Pro running OS X Snow Leopard (10.7.4) with a 2.2 GHz Intel Core i7 processor and 8GB of memory.

5.2 Results

Table 2 shows the results of the performance evaluation. The second column specifies the size of the Pareto Front, that is, the number of Pareto-optimal solutions (for unspecialized models). The third column gives the median times (over 20 repetitions) to compute these solutions. It turns out that all models have a small number of Pareto-optimal solutions. For models with a single objective, multiple solutions arise when each optimal solution achieves the same total quality attribute value via different design decisions. The fourth column shows the median size of the Pareto Front sizes for the 20 randomly-generated satisfiable partial configurations; the last column shows the corresponding median time to compute them. As expected, partial configurations constrain the solution space, also reducing the

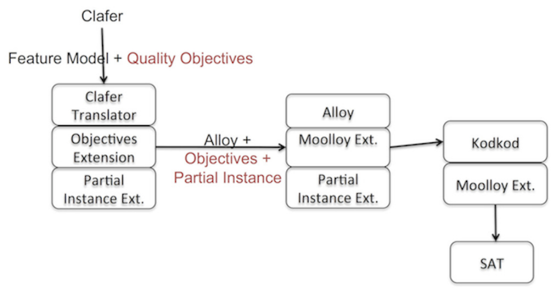


Figure 6: ClaferMoo architecture overview

Table 1: Software Product Lines and Attributed Feature Models

Software Product Line	Features	Cross-Tree Constraints	Quality Attributes
Apache	9	1	Performance
Berkeley Db I	9	0	Footprint
Berkeley Db II	12	0	Footprint, Price, Reliability
LinkedList	18	1	Footprint
PKJAB	11	0	Footprint
Prevayler	5	1	Footprint
SQLite	85	2	Footprint
UML Violet	100	89	Footprint
ZipMe	8	0	Footprint

size of the Pareto Front. The computation times are also well-behaved—more constrained problems require less time.

In general, increasing the number of features appears to lead to increased solving time, and, in the case of UML Violet, which timed-out in the experiment, the number of cross-tree constraints may also have lead to increased solving times. With the exception of SQLite and UMLViolet, the single-objective models are solved in a few seconds or less. The multi-objective model, Berkeley Db II, is solved in a few minutes or less.

6. THREATS TO VALIDITY

The main threat to the external validity of the evaluation is the small sample of models. Feature models attributed with quality attributes are rare; we have used models based on recent work on extracting such models from variability-rich software [14]. The majority of these models are small and only one has multiple quality attributes, and no additional multi-objective attributed feature models were available from [14] [12] [13]. On the other hand, the selected models describe real-world software of sufficient complexity such that tool support is needed to reason about their variants.

7. RELATED WORK

We are unaware of any prior work of applying multi-objective optimization in the context of variability models.

A series of papers—[14], [12], [13], and [11]—by the research group of Sven Apel have presented a methodology for quantifying quality attributes (non-functional properties) in software product lines, and recommended that these models could be used for optimization. They built a special tool called SPL Conqueror to measure the footprint, performance, *etc.*, of different variants of a software product line, and to infer attributed feature models. In contrast, the work presented in this paper makes use of such models to help stakeholders automatically complete the configuration process by finding the optimal variant, respecting the choices already made by stakeholders.

Table 2: Running times for Optimizing Software Product Lines for Quality Attributes

SPL	Pareto Front Size	Time to Compute Pareto Front	Pareto Front Median Size for Partially Configured SPL	Median time to compute PF for Partially Configured
Apache	2	0.2 s	1	90 ms
Berkeley DB	1	0.8 s	1	0.4 s
Berkeley DB II	6	3.8 min.	2.5	46 s
LinkedList	2	4.6 s	1	0.7 s
PKJAB	2	0.1 s	1	78 ms
Prevayler	1	0.1 s	1	59 ms
SQLite	N/A	>11.1 min.	N/A	>18 min.
UML Violet	Time Out	N/A	N/A	N/A
ZipMe	2	0.1 s	1.5	50 ms

In [6], Sincero *et al.* introduce quality attributes into the software configuration process by giving feedback to stakeholders about their feature choices, attempting to predict the values of quality attributes. They test their approach with a subset of the Linux kernel, and they modify the Linux configuration tool to show the impact of feature selection on quality properties of the kernel.

The work by Benavides *et al.* [2] introduces extended feature models, which are translated to a constraint solver in order to find optimal solutions; however, they only use single-objective optimization (modelling multiple objectives by weighting). They implement their approach using the commercial constraint solver OPL-Studio.

In [18], White *et al.* present a fast (polynomial time) approximation algorithm to select near-optimal products configurations using a single objective while respecting multiple resource constraints. They test their approach on randomly generated feature models and show that models having up to 10000 features can be solved in a couple of seconds, and that their solutions are within 90% of the optimal products.

8. CONCLUSION AND FUTURE WORK

Versatile software inevitably leads to variability, and companies using product line engineering can benefit from exploring the design space of optimally configured product lines. The ClaferMoo tool can help stakeholders to incorporate their desired quality attributes into the configuration process for software product line or customizable software. This can be achieved by specifying goals, using new syntax in Clafer, over the quality attributes, and by allowing a multi-objective constraint solver to enumerate the optimal products.

Future work is needed in order to study how the performance of the solver is impacted by characteristics of the attributed feature models, such as number of features, number of cross-tree constraints, and feature interactions. Additionally, a user-study is needed in order to evaluate the efficacy of the approach taken for specifying attributed feature models with goals in Clafer. As well, product line engineers using multi-objective optimization need to identify strategies for selecting optimally configured products, given the various trade-offs among optimal configurations in the presence of multiple conflicting design goals. And most importantly, future work needs to address the scalability limitations of the current optimization infrastructure.

By applying this approach to attributed feature models for both academic and open source software product lines from different domains, it has been shown that there is value in using such an approach, that, with the exception of one of the models, the multi-objective optimization solver can easily compute. There is much promise in further exploring this research.

References

[1] K. Bąk, K. Czarnecki, and A. Wasowski. Feature and meta-models in clafer: Mixed, specialized and coupled. In *3rd International Conference on Software Language Engineering*, pages 291–301, October 2010.

[2] D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated reasoning on feature models. In *Advanced Information Systems Engineering: 17th International Conference*, LNCS. Springer, 2005.

[3] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki. Variability modeling in the systems software domain. Technical report, Generative Software Development Laboratory, University of Waterloo, 2012.

[4] P. C. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, 2001.

[5] D. Jackson. *Software Abstractions: Logic, Language and Analysis*. The MIT Press, revised edition, 2012.

[6] W. S.-P. Julio Sincero and O. Spinczyk. Approaching non-functional properties of software product lines: learning from products. In *Proceedings of the 2010 Asia Pacific Software Engineering Conference*, APSEC '10. IEEE, 2010.

[7] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Software Engineering Institute - CMU, 1990.

[8] V. Montaghami and D. Rayside. Extending Alloy with Partial Instances. In *ASM, Alloy, B, VDM and Z (ABZ)*, June 2012.

[9] K. Pohl, G. Böckle, and F. J. v. d. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. ISBN 3540243720.

[10] D. Rayside, H.-C. Estler, and D. Jackson. A Guided Improvement Algorithm for Exact, General Purpose, Many-Objective Combinatorial Optimization. Technical Report MIT-CSAIL-TR-2009-033, MIT Computer Science and Artificial Intelligence Laboratory, 2009. URL `http://hdl.handle.net/1721.1/46322`.

[11] N. Siegmund, M. Rosenmuller, M. Kuhlemann, C. Kastner, and G. Saake. Measuring non-functional properties in software product lines for product derivation. In *VAMOS*, pages 1–31, June 2010.

[12] N. Siegmund, M. Rosenmuller, C. Kastner, P. G. Giarrusso, S. Apel, and S. S. Kolesnikov. Scalable prediction of non-functional properties in software product lines. In *Software Product Line Conference (SPLC), 2011 15th International*, pages 160 –169, August 2011.

[13] N. Siegmund, M. Rosenmuller, M. Kuhlemann, C. Kastner, S. Apel, and G. Saake. Spl conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, 1(3):1–31, June 2011.

[14] N. Siegmund, , S. Kolesnikov, C. Kastner, S. Appel, D. Batory, M. Rosenmuller, and G. Saake. Predicting performance via automated feature-interaction detection. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, New York, NY, USA, 2012. ACM.

[15] R. Steuer. *Multiple Criteria Optimization: Theory, Computations, and Application*. John Wiley & Sons, Inc., New York, 1986. ISBN 0-471-88846-X.

[16] S. T. Stewart. Extending Alloy with Multi-Objective Optimization. Technical report, University of Waterloo, 2012.

[17] E. Torlak and G. Dennis. Kodkod for Alloy Users. In *Proceedings of the First Alloy Workshop*, Portland, OR, USA, 2006.

[18] J. White, B. Dougherty, and D. C. Schmidt. Selecting highly optimal architectural feature sets with filtered cartesian flattening. *Journal of Systems and Software*, 82(8):1268–1284, Aug. 2009.