Cardinality-Based Feature Models With Constraints: A Pragmatic Approach

Clément Quinton, Daniel Romero, Laurence Duchien

Cardinality-Based Feature Models With Constraints: A Pragmatic Approach

Clément Quinton
INRIA Lille - Nord Europe
LIFL UMR CNRS 8022
University Lille 1, France
clement.quinton@inria.fr

Daniel Romero
INRIA Lille - Nord Europe
LIFL UMR CNRS 8022
University Lille 1, France
daniel.romero@inria.fr

Laurence Duchien
INRIA Lille - Nord Europe
LIFL UMR CNRS 8022
University Lille 1, France
laurence.duchien@inria.fr

ABSTRACT

Feature models originating from Software Product Line Engineering are a well-known approach to variability modeling. In many situations, the variability does not apply only on features but also on the number of times these features can be cloned. In such a case, cardinality-based feature models are used to specify the number of clones for a given feature. Although previous works already investigated approaches for feature modeling with cardinality, there is still a lack of support for constraints in the presence of clones. To overcome this limitation, we present an abstract model to define constraints in cardinality-based feature models and propose a formal semantics for this kind of constraints. We illustrate the practical usage of our approach with examples from our recent experiences on cloud computing platform configuration.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design—*Methodologies, Representation*; D.2.13 [**Software Engineering**]: Reusable Software—*Domain engineering, Reuse models*

General Terms

Variability, Modeling

Keywords

Cardinality, Feature Model, Constraint

1. INTRODUCTION

In *Software Product Line Engineering* (SPLE), the definition of variabilities and commonalities, known as variability modeling, is a central activity [4, 12]. A well-known approach to variability modeling is by means of *Feature Model* (FM) [2]. Despite their widespread use, conventional FMs are sometimes not sufficient to describe variabilities, which imposes a need for a richer model. In our recent work and experiences in the configuration of cloud computing platforms using SPL techniques [13, 14], we had to go one step further in feature modeling by specifying the number of times a feature can be included in the configuration. This operation is described in the literature as feature *cloning* and is handled by cardinality-based FMs. However, existing constraints (*implies* and *excludes*) defined in terms of features suffer from a lack of expressiveness in the presence of clones [10].

In this paper, we report on our recent experiences on cloud computing platform configuration to highlight the needs of expressing complex constraints for cardinality-based feature modeling. The contribution of this paper is twofold. First, we analyze through a motivating example what is required to express constraints over the set of feature clones. Second, we define an abstract model for cardinality-based FMs and constraints and describe how it can be used on top of existing solvers to develop a tool-based support. We illustrate the practical usage of our approach through case studies dealing with the configuration of cloud computing platforms.

The paper is organized as follows. In Sec. 2, we discuss the motivation and describe the challenges to tackle. The proposed approach is defined in Sec. 3. In Sec. 4, we describe close-related work. Finally, Sec. 5 concludes the paper and presents the perspectives of our work.

2. MOTIVATION & CHALLENGES

In this section, we report on our experience with the deployment of applications in the cloud and the configuration of cloud platforms. We then identify several challenges related to cardinality-based constraints modeling.

2.1 Motivating Example

In previous work [13, 14], we described the way SPLE techniques can be used to configure applications to be deployed on the cloud as well as cloud platforms themselves. In the cloud computing paradigm, computing resources are delivered as services. Such a model is usually described as *Anything as a Service* (XaaS or *aaS), where *anything* is divided into layers from *Infrastructure* to *Software* including *Platform*. This model in layers offers many configuration and dimension choices [9]. For example, configuring a PaaS requires the selection of the database and application server type, the configuration of security concerns or the definition of required compilation tools or libraries, as depicted by the `Jelastic`[1] PaaS FM, Fig. 1.

[1] `https://app.jelastic.com//`

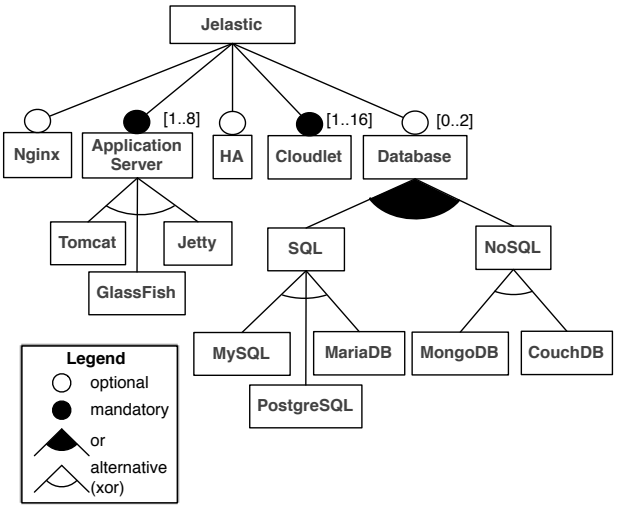

Figure 1: Jelastic cloud feature model (excerpt)

We used cardinality-based FMs to describe such a variability. However, existing approaches lack in expressiveness for feature cardinalities and constraints over the set of clones of a feature. For example, regarding FIG. 1, the `Nginx` feature is actually a load balancer. This means that if there is at least 2 `Application Servers` configured, the `Nginx` load balancer must also be configured. This constraint can not be written `Application Server` → `Nginx` with classical implication constraints. Another example we met was: *"All virtual machines hosting more than one application server require at least* 3 *acceptors"*, where a relationship occurs between clones.

In such cases, constraints do not deal with features and have to be expressed in terms of clones. In particular, these constraints must provide means to define quantifier over the set of feature clones.

2.2 Challenges

Our experience in the deployment of cloud applications and the configuration of cloud platforms provides evidence that support for expressing constraints and reasoning about cardinalities can significantly improve the way cardinality-based FMs are developed and evolved. Therefore, our approach should address, at least, the following challenges:

$\mathcal{C}_1$: *Expressing constraints with feature cardinality support.* Constraints must be expressed in terms of clones, *e.g.*, by specifying that clones of one feature should be included or excluded from a product. Moreover, quantifiers should be used over the set of clones for a feature, *e.g.*, to specify that a feature requires a given amount of clones of another feature [10].

$\mathcal{C}_2$: *Reasoning about cardinality-based FMs.* Once the feature model specified, *i.e.*, constraints with quantifiers over the set of clones properly expressed, developers need a tool for reasoning about FMs. Such reasoning includes the selection or deselection of features and the validation of configurations.

3. CARDINALITY-BASED CONSTRAINTS

We propose to use an abstract syntax to tackle the aforementioned challenges. We thus propose an implementation-independent model defined as a metamodel. We describe in the following the metamodel details and give some examples on the way we use this metamodel to define cardinality-based constraints.

Proposed Metamodel

The metamodel we describe in this paper is an extension of the feature metamodel proposed by Parra *et al.* [11]. Due to space limitations, we only describe the part regarding cardinality-based constraints, but you can find the whole of it in [1]. The metamodel defines three kinds of constraints, as depicted by FIG. 2. Two of them are well-known constraints, `Implies` and `Excludes`, where the selection of a feature in a product implies the selection of another feature and excludes another feature from the product, respectively.

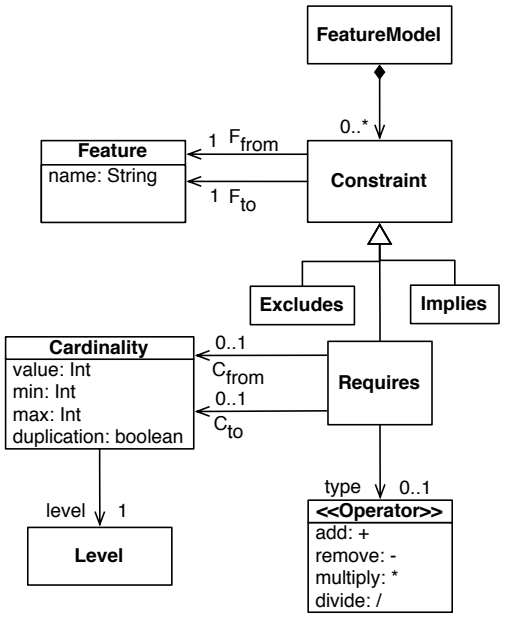

Figure 2: Cardinality-based constraint metamodel

However, in clone-enabled FMs, these constraints, expressed in terms of features, are not expressive enough. Furthermore, one should be able to specify quantifiers over the set of clones of certain features. Therefore, we propose a third kind of constraint called *requires*.

In order to introduce our *requires* constraint, we take as example the Jelastic PaaS. In particular, we use the following specifications from the Jelastic documentation[2] that have to be expressed in the Jelastic FM (see FIG. 1):

(s_1): One cloudlet is allocated per database instance.

(s_2): High Availability (HA) provides session replication for application servers.

[2] `http://jelastic.com/docs`

When looking into the details, these specifications can be seen as constraints to express that *(cons$_1$): each database instance requires one cloudlet* and *(cons$_2$): when HA is selected, the number of application servers is multiplied by two*, since each instance is replicated.

The "Requires" Constraint

In the previous example, the ($cons_1$) constraint can not be expressed *Database* → *Cloudlet* for two reasons. First, the meaning is not clear enough since this constraint is expressed in terms of features. The constraint can be interpreted as *a Database implies all clones of Cloudlet*. Second, the configured clone of cloudlet can not be shared among databases since each database requires one cloudlet. Thus, while selecting a second database in the product configuration, one may think that the constraint is already satisfied since a cloudlet has already being selected with the first database. On the other hand, considering the *(s_2)* specification, we can not use `Implies` constraint to express the multiplicity. We need then some mechanisms employing quantifiers and operators to define ($cons_2$). In order to express this kind of constraints, we propose the *requires* constraint defined as follows:

Definition 1. (Requires Constraint)
A *requires* constraint Req_{cons} is written $[C_{from}]\ F_{from} \rightarrow \delta\ [C_{to}]\ F_{to}$, where

- $F_{from}, F_{to} \in \mathcal{F}$ where *(i)* $\mathcal{F}$ is the non empty set of features of the FM and *(ii)* $F_{from} \neq F_{to}$;
- C_{from}, C_{to} are cardinalities. Each one defines an interval [i-j] with i, j $\in \mathbb{N}$ and i $\leq$ j. C_{from} (respectively C_{to}) is the quantifier over the set of clones of F_{from} (respectively F_{to}) ;
- $\delta \in \{\varnothing, +, -, *, /\}$ and $\texttt{op} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, where $\texttt{op}(a, b) = a\ \delta\ b$ with $\delta \neq \varnothing$;
- $\omega : \mathcal{F} \rightarrow \mathbb{N} \times \mathbb{N}$ indicates the cardinality of each feature. $\forall f \in \mathcal{F},\ \omega(f) = [n, m]$ where n is the lower bound and m the upper bound of the cardinality.
- The operator δ, the lower bound, the upper bound and the whole cardinality are optional.

The *requires* constraint denotes that,

- if $\delta \neq \varnothing,\ \omega(F_{from}) \subseteq C_{from} \Rightarrow \texttt{op}(\texttt{card}(F_{to}), C_{to})$, where $\texttt{card}(f)$ returns the number of feature f instances, with $n \leq \texttt{card}(f) \leq m$.
- else, $\omega(F_{from}) \subseteq C_{from} \Rightarrow \omega(F_{to}) \subseteq C_{to}$.

The previous definition is reified in the metamodel with the `Requires` meta-class. This constraint applies *from* a feature *to* another one and specifies which `Operator` should be used, if any. However, cardinalities can also be specified and applied on features affected by the *requires* constraint. In such a case, the `Cardinality` meta-class has a completely different meaning than the one we have regarding feature and group feature cardinality. To illustrate this, we use our scenario with constraints.

Fig. 3 (a) represents the specification *(s_1)* using our *requires* notation. Since each database instance requires one cloudlet, the cloudlet cardinality is incremented by one. In such a case, the lower bound of the *to* cardinality is used, and one can choose to set the upper bound or not. In this example, there is no cardinality specified for the *from* feature, and there is no upper bound for the *to* feature cardinality. In Fig. 3 (b), the first constraint goes the same way, but the value and the operator are different. Such constraint represents the specification *(s_2)*. The second constraint of Fig. 3 (b) presents how the cardinality can be applied over the set of feature clones. This constraint describes the fact that *if there is at least two configured Application Servers, then the Nginx load balancer is required*. The *from* cardinality is used and a value, here 2, is assigned to the *min* attribute whereas the *max* attribute is not used.

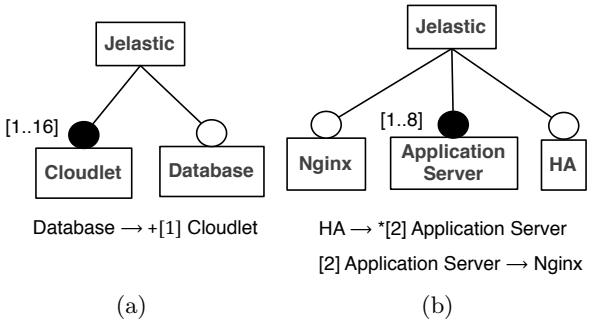

Figure 3: Clonable features and associated constraints

This means that if there is *at least* two application servers, whatever the maximum number of instances[3], the configured product requires a Nginx load balancer. Let us now take as example this made-up constraint: *if there is at least two configured Application Servers and at most four, then at most eight Cloudlets are required*. Relying on our proposed metamodel, it can be expressed as follows:

[2,4] Application Server → [0,8] Cloudlet

where the min and max attributes of the C_{from} and C_{to} cardinalities have a value, respectively 2, 4 and 0, 8. In this case, the 0 value indicates that the constraint applies whatever the value of the C_{to} lower bound as long as the upper bound does not exceed 8. This *[0,8]* cardinality also distinguishes with the *[8]* cardinality, whose meaning is *at least 8*.

Configuration Concerns

Actually, our approach distinguishes between two kinds of cardinality-based constraints: *configuring constraints* and *checking constraints*. The former is used when configuring the product and affects the number of clones of a given feature, by defining a value combined with an operator, *e.g.*, *Database → +[1] Cloudlet*. The latter is used when the configuration is over, to check *(i)* whether the number of configured clones is correct or not, *e.g.*, *[2,4] Application*

[3]Assigning a value to the *max* attribute is not mandatory, since it can not exceed the feature cardinality upper bound, here 8.

Server → [0,8] Cloudlet, and *(ii)* if this number of clones is counted as it should be. Indeed, the number of configured clones can be correct regarding constraints but it also has to be correct regarding the *scope* and *level* when modeling the FM. In our approach, we rely on the scope and level concepts as it was presented by Michel *et al.* [10]. We illustrate this situation with the two products depicted by FIG. 4.

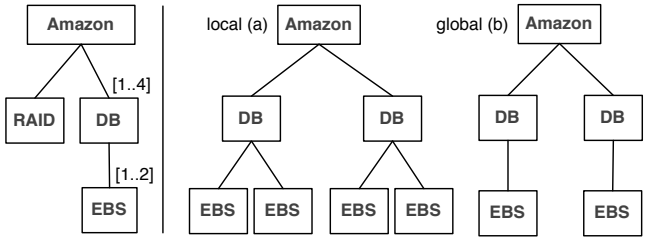

Figure 4: Examples of product according to the specified level

The `Amazon` IaaS cloud provides storage support by reliying on `EBS`[4] storage volumes that can be attached to a running Amazon EC2 instance requiring a database `DB`. For better recovery and failure management, `RAID` must be selected if there are at least two `EBS` volumes, thus ensuring data replication among these volumes, as expressed by the following constraint: *[2] EBS → RAID*. In both FIG. 4 (a) and FIG. 4 (b) cases, this constraint has to be satisfied since each product includes at least two clones of `EBS`. If the level related to the `Database` feature cardinality is local, then the constraint is semantically valid in the case (a), since each clone of `Database` can have up to two clones of `EBS`. In contrast, if the level is global, the constraint is semantically respected in the case (b), since all clones of `EBS` are shared among the whole product. Thus, each clone of `Database` has one clone of `EBS`, reaching the maximum number of clones allowed, 2. In such a situation, the C_{from} cardinality level must be set to describe in which way clones must be count.

Tool Support

Relying on the previously described metamodel, we developed a configurator with support for feature cardinalities. This configurator is part of the SALOON framework [13] that provides an API to configure and manage cardinality-based FMs. More details are given about this API in [1]. Cardinality-based FMs and their constraints are graphically defined as instances of the proposed metamodel by using the *Eclipse Modeling Framework* (EMF) [16], which is one of the most widely accepted metamodeling technologies. Although we could provide a full domain specific language to handle both FM modeling and reasoning, we believe that this separation has several benefits. First, a graphical editor favors readability, usability and productivity when dealing with FM modeling. Indeed, it is quite error-prone to syntactically define cardinality-based FMs by hand whereas a GUI is very helpful. That said, it remains possible to use the EMF text editor to define FM models with an XML-based syntax. Second, the modeled FM can be easily handled by existing tools to add more capabilities. For example, one can model assets related to features using our metamodel (not described in this paper because out of scope). Then, generation tools such as Acceleo[5] are used to visit the model and generate the expected configured product.

Once cardinality-based FMs are modeled, SALOON provides rules to translate these FMs into constraints that are handled by off-the-shelf CSP solvers to reason about the FMs, *e.g.*, calculating the number of products or checking the validity of a configuration. In particular, SALOON relies on the well-known Java CSP Choco solver[6]. Each feature of the FM is thus associated to a variable in the CSP, and each variable holds as domain the cardinality related to the feature. For example, feature `DB` cardinality in FIG. 4 is translated as $DB \in \{1, 4\}$. Translation from FMs to CSP constraints has already been studied in the literature. We rely on the translation rules described by Benavides *et al.* [3] to perform operations on FMs using Choco. Regarding the *requires* constraint, the following constraint

$$[2, 4]A \rightarrow [0, 8]B$$

is translated into CSP the following way

$$implies(and(2 \leq A\ ;\ A \leq 4); and(0 \leq B\ ;\ B \leq 8)).$$

In Short

We proposed a metamodel and we described how it can be used to define constraints in cardinality-based FMs, thus facing the challenge $\mathcal{C}_1$. In particular, we provided a new constraint definition, *requires*, that enables the specification of operators and quantifiers over the set of clones of certain features to be included in the product. We also introduced SALOON, a framework handling such FMs by relying on a CSP solver, and described how cardinality-based constraints are translated into CSP constraints, thus facing the challenge $\mathcal{C}_2$.

4. RELATED WORK

Cardinality-based FMs were first introduced in [15] where UML *multiplicities* were used as an extension to the original FODA notation to support feature cloning. The authors introduced the notion of feature cardinality and proposed a syntax to define this cardinality in the FM. Our approach is based on this syntax, as the following approaches do.

Czarnecki *et al.* [5] proposed a metamodel for cardinality-based FMs and formalized the group cardinality notion, giving a new semantics to the existing cardinality. These authors went further in their research by proposing to use languages like OCL to define constraint for FMs with cardinalities [6]. We pursue the same goal but we extend the support for cardinalities semantics to constraints and provide a way to express complex constraints over the set of feature clones regarding this semantics, what is not feasible with OCL.

Zhang *et al.* [17] presented a BDD based approach to verify constraints with cardinalities, based on their own semantics of cloning. They described their different constraint patterns and the way they can be verified but did not provide any abstract syntax or tool support to define such FDs. In [8], the authors presented their own metamodel and the way they rely on model-driven engineering to configure their FMs.

[4] *Elastic Block Store*, `http://aws.amazon.com/ebs/`

[5] `http://www.acceleo.org/`

[6] `http://www.emn.fr/z-info/choco-solver/`

They also introduce a new kind of constraint denoted as *Use* where a feature A can use a given amount of features B. The approach we propose in this paper goes in the same direction, but we go further in cardinality-based constraints support, in particular regarding quantifiers and semantics.

Michel *et al.* [10] precisely defined the syntactic and semantic domain of cardinality-based FMs. Moreover, they defined the semantics of cloning, regarding group cardinalities and feature ones. However, they do not provide any support for cross-tree constraints over the set of clones of a feature. Finally, Dhungana *et al.* [7] propose an approach based on generative constraint satisfaction problems to handle constraints in the context of cloned features. They also introduce a notation to express quantifiers in constraints over the set of cloned features, as we do in this paper. Contrarily to them, we propose an abstract model to define such constraints that can be easily implemented in the wished syntax.

5. CONCLUSION & FUTURE WORK

In this paper, we present an approach to manage constraints in cardinality-based feature models. Based on case studies dealing with the configuration of cloud computing platforms, we describe an abstract model to manage such constraints. We illustrate the benefits of using such a model with real constraints and provide a formal semantics. We then show how FMs described using our metamodel can be translated into constraints that can be handled by off-the-shelf CSP solvers to reason about the FMs. This translation process is part of the SALOON framework. Our approach thus faces the challenges identified in SEC. 2.2 regarding expressing and reasoning about constraints in cardinality-based FMs.

The next step of our work is to empirically assess our approach. Although based on several case studies, the approach needs to be validated among substantial numbers of other cardinality-based FMs. We are currently defining FMs from several cloud providers to validate the model and API on varied FMs and kind of constraints. This will help us identify complementarity ways of managing cardinality-based FMs and improve our tool support. At the time of writing, we develop several translation operators to make our API as complementary as possible with existing approaches by providing support for different formats.

Acknowledgments

This work is supported by Ministry of Higher Education and Research, Nord–Pas de Calais Regional Council, the FEDER through the *Contrat de Plan Etat Region Campus Intelligence Ambiante* (CPER CIA) 2007-2013 and the EU FP7 PaaSage project.

6. REFERENCES

[1] `http://researchers.lille.inria.fr/~cquinton/splc/splc2013.html`.

[2] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated Analysis of Feature Models 20 Years Later: A Literature Review. *Inf. Syst.*, 35(6):615–636, Sept. 2010.

[3] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. Using Java CSP Solvers in the Automated Analyses of Feature Models. In *Proceedings of the 2005 international conference on Generative and Transformational Techniques in Software Engineering*, GTTSE'05, pages 399–408, 2006.

[4] P. Clements and L. M. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[5] K. Czarnecki, S. Helsen, and U. W. Eisenecker. Formalizing Cardinality-based Feature Models and their Specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.

[6] K. Czarnecki and C. H. P. Kim. Cardinality-Based Feature Modeling and Constraints: A Progress Report. In *International Workshop on Software Factories at OOPSLA'05*. ACM, 2005.

[7] D. Dhungana, A. Falkner, and A. Haselbock. Configuration of Cardinality-Based Feature Models Using Generative Constraint Satisfaction. In *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, pages 100 –103, 30 2011-sept. 2 2011.

[8] A. Gómez and I. Ramos. Cardinality-Based Feature Modeling and Model-Driven Engineering: Fitting them Together. In *Fourth International Workshop on Variability Modelling of Software-Intensive Systems*, pages 61–68, 2010.

[9] P. Mell and T. Grance. The NIST Definition of Cloud Computing. Technical report, National Institute of Standards and Technology, 2009.

[10] R. Michel, A. Classen, A. Hubaux, and Q. Boucher. A Formal Semantics for Feature Cardinalities in Feature Diagrams. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS'11, pages 82–89, 2011.

[11] C. Parra, A. Cleve, X. Blanc, and L. Duchien. Feature-based composition of software architectures. In *Proceedings of the 4th European conference on Software architecture*, ECSA'10, pages 230–245, 2010.

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

[13] C. Quinton, N. Haderer, R. Rouvoy, and L. Duchien. Towards Multi-Cloud Configurations Using Feature Models and Ontologies. In *1st International Workshop on Multi-Cloud Applications and Federated Clouds*, Prague, Czech Republic, Apr. 2013.

[14] C. Quinton, R. Rouvoy, and L. Duchien. Leveraging Feature Models to Configure Virtual Appliances. In *Proceedings of the 2nd International Workshop on Cloud Computing Platforms*, CloudCP '12, pages 2:1–2:6. ACM, 2012.

[15] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow. Extending Feature Diagrams with UML Multiplicities. In *6th World Conference on Integrated Design & Process Technology (IDPT2002)*, June 2002.

[16] D. Steinberg, et al. *EMF: Eclipse Modeling Framework (2nd Edition)*. 2nd revised edition, 2009.

[17] W. Zhang, H. Yan, H. Zhao, and Z. Jin. A BDD-Based Approach to Verifying Clone-Enabled Feature Models' Constraints and Customization. In H. Mei, editor, *High Confidence Software Reuse in Large Systems*, volume 5030 of *Lecture Notes in Computer Science*, pages 186–199. 2008.