Using Decision Rules for Solving Conflicts in Extended Feature Models

Lina Ochoa Oscar González-Rojas

Systems and Computing Engineering Department, School of Engineering, Universidad de los Andes, Bogotá, Colombia
{lm.ochoa750, o-gonza1}@uniandes.edu.co

Thomas Thüm

Institute of Software Engineering and Automotive Informatics, Technische Universität Braunschweig, Brunswick, Germany
t.thuem@tu-braunschweig.de

Abstract

Software Product Line Engineering has introduced feature modeling as a domain analysis technique used to represent the variability of software products and decision-making scenarios. We present a model-based transformation approach to solve conflicts among configurations performed by different stakeholders on feature models. We propose the usage of a domain-specific language named *CoCo* to specify attributes as non-functional properties of features, and to describe business-related decision rules in terms of costs, time, and human resources. These specifications along with the stakeholders' configurations and the feature model are transformed into a constraint programming problem, on which decision rules are executed to find a non-conflicting set of solution configurations that are aligned to business objectives. We evaluate CoCo's compositionality and model complexity simplification while using a set of motivating decision scenarios.

Categories and Subject Descriptors D.2.13 [*Reusable Software*]: Domain engineering; D.2.9 [*Management*]: Software configuration management

Keywords Domain engineering, extended feature model, conflicting configurations, domain-specific language, model transformation chain, constraint satisfaction problem

1. Introduction

Feature models have been used during the last decades as an important domain engineering modeling concept, which provides an abstract, explicit, and concise representation of a system's variability [7]. A feature model defines a Software Product Line (SPL) for a given domain [14], and it also represents a set of decisions in a decision-making process. In this scenario, the set of selected and deselected features correspond to a set of decisions defined by a stakeholder. Usually, there are multiple stakeholders that participate on the same decision-making context. Thus, different roles such as business leads, IT leads, project managers, among others, base their decisions on their knowledge and expertise. However, the decision products defined by these stakeholders can result into different conflicts between configurations and business needs.

Some approaches manage conflicts by avoiding the introduction of inconsistencies, while using a decision propagation strategy [6]. Other approaches focus on creating configurations by allowing inconsistencies to be solved *a-posteriori* [16]. Both approaches are supported with techniques like Binary Decision Diagrams (BDDs), Satisfiability (SAT) solvers [4] [17] [20], and Constraint Satisfaction Problem (CSP) solvers [5] [22]. In addition, there is no such approach that considers the use of quantitative decision rules to converge stakeholders' points of view into the best product (*i.e.,* decision scenario). These approaches neither prioritize business needs nor stakeholders expertise. The evaluation of additional decision rules associated to different domains such as costs, time, or human resources are required to manage conflicts.

This paper presents a new approach to solve conflicts between stakeholders' configurations performed over feature models (*cf.* Figure 1). First, we propose a Domain-Specific Language (DSL) named CoCo to specify non-functional attributes that define business analysis dimensions (*i.e.,* costs, time, and human resources). IT and business stakeholders can configure decision products by selecting a set of valid feature combinations in the feature model. CoCo also allows the specification of business-related constraints (namely decision rules) in terms of the above analysis dimensions to scope the set of potential non-conflicting solutions among configurations. These solutions are aligned to concrete busi-

SLE'15, October 26–27, 2015, Pittsburgh, PA, USA

http://dx.doi.org/10.1145/2814251.2814263

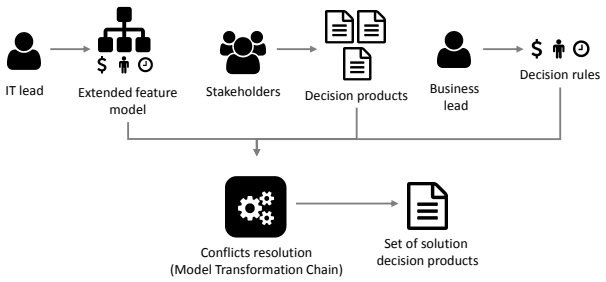

Figure 1: Resolution of conflicts on decision products.

ness objectives. Second, we present a model-based transformation approach that combines the feature model, the stakeholders' configurations, and the defined decision rules to generate a Constraint Satisfaction Problem (CSP). The resulting model offers stakeholders a set of non-conflicting products that satisfies the business needs. We designed tool support for CoCo in a compositional manner. That is, we can use a state-of-the-art tool for feature modeling and CSP solving as-is by defining attributes and decision rules in an independent format. Finally, we evaluate the compositionality of CoCo specifications and model complexity simplification by using a set of predefined decision scenarios.

This document is organized as follows. Section 2 presents a set of concepts that will be used along the paper. Section 3 presents a feature-oriented decision model as sample scenario to understand the domain concepts and the open problems for dealing with conflicting configurations. Section 4 describes the main elements of the proposed DSL to specify feature attributes and decision rules for solving conflicts. Section 5 presents the model transformation chain created to implement the defined strategy into an executable CSP. Section 6 illustrates two experiments performed to evaluate our approach. In Section 7, we discuss related work. Finally, conclusions and future work are presented.

2. Conflicting Configurations: Core Concepts

In this section, concepts such as feature modeling, configurations, model transformation chains, and constraint satisfaction problems are briefly introduced.

Feature modeling. *Software Product Line Engineering* (SPLE) is a paradigm related to software engineering, used to define and derive similar software products from reusable core assets and the understanding of a particular domain [14]. Therefore, SPLE relies on the identification of variabilities and commonalities among the characteristics or features of a product family which can be represented through a feature model [11].

A *feature model* is composed of a feature diagram and cross-tree constraints. A *feature diagram* is a tree-based structure where nodes represent features, which are characteristics or properties that can be associated to a software product. Definition 1 illustrates the formalization of this concept based on the Schobbens et al. [18] proposal. Features are initialized by defining a boolean state. Therefore, if the feature is selected it will have a *true* value, and if it is deselected it will have a *false* value. In addition, features can be classified as atomic features which are situated at the leaf level of the diagram, otherwise they are classified as non-atomic features and are used as grouping nodes [19]. Furthermore, feature diagrams also contain a set of tree constraints, and integrity constraints.

DEFINITION 1. *A* feature diagram FD *that represents a feature model is defined as* $FD = (N, r, TC, IC, CTC)$*, where:*

- N *represents the complete set of features.*
- r *represents the root feature* ($r \in N$)*.*
- TC *represents the tree constraints set* ($|N| - 1 \leq |TC|$)*.*
- IC *represents the integrity constraints set.*
- CTC *represents the cross-tree constraints set.*

Tree constraints define different relationships among parent features and children features: *alternative* relations (or-exclusive functions) represent that exactly one child feature has to be selected when the parent is selected; *or* relations which represent the needed selection of at least one feature from the children features subset, and at most all elements of this subset; *mandatory* relations that enforce the selection of a given feature; and *optional* relations that allow both the selection or deselection of the given feature. *Integrity constraints* define two types of relationships among any two or more features: *requires* where given a set of selected features the selection of a different set of features is enforced; and *excludes* where given a set of selected features the deselection of a different set of features is also enforced. Finally, *cross-tree constraints* define logical formulas in order to express relations that extend the integrity constraints expressiveness.

Furthermore, feature models can be extended by adding non-functional properties as feature attributes. In that way, features will have both a selection state and a set of feature attributes.

Configurations. A *configuration* is defined as a set of selected and deselected features from a feature model [8]. A configuration can be classified as valid or invalid, as well as partial or complete. A *valid* configuration conforms to or do not violate tree constraints, integrity constraints, and cross-tree constraints related to the feature model, otherwise it is *invalid*. In addition, a configuration is *complete* when each feature of the model (both atomic and non-atomic) has a defined selection state, otherwise it is a *partial* configuration. Thus, a *product* is defined as a configuration that is both valid and complete [1].

By its side, stakeholders can define a set of products from a feature model during a configuration process. A conflict among configurations raises when configurations are valid but between them are some exclusive decisions defined by

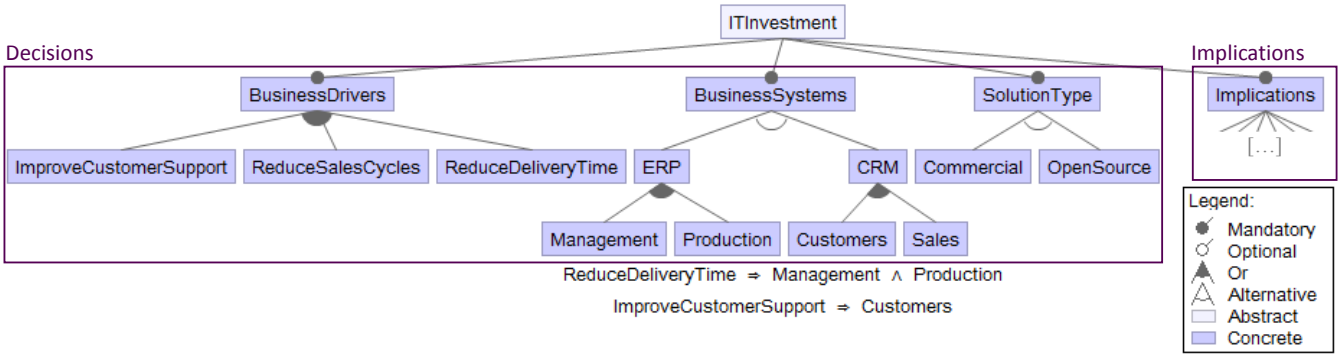

Figure 2: IT investment feature model.

the feature model constraints. In that way, the selection of one feature in a specific stakeholder configuration could be exclusive with a different selection made by a different stakeholder (*cf.* Definition 2).

DEFINITION 2. *Consider a set of valid configurations* $C_S = \{C_1, ..., C_j, ..., C_m\}$, $\forall j : 1 \leq j \leq m$. *Now, consider a resulting configuration* C_R *from the union operation over all configurations at* C_S *(i.e.,* $C_R = \bigcup_{j=1}^{m} C_j$*). Then, a* conflict among configurations *arises if and only if* C_R *is not valid.*

Model transformation chain. A *Model Transformation Chain* (MTC) is a sequence of transformation tasks that takes a high-level input model and translates it into low-level output assets. In that way, problem domain concepts are transformed into solution domain concepts [24].

Constraint programming. *Constraint programming* is a paradigm based on establishing relationships among a set of variables. A CSP is obtained when problems are defined in terms of a finite set of variables and constraints over those variables.

DEFINITION 3. *A* CSP *is a triple* (V,D,C) *where* $V = \{v_1, v_2, ..., v_n\}$ *is the problem variables finite set,* $D = \{d_1, d_2, ..., d_n\}$ *is the respective variables domain set such that* $\forall i : v_i \in d_i$ *and* $1 \leq i \leq n$*, and* $C = \{c_1, c_2, ..., c_n\}$ *is the problem constraints set [1].*

CSP objectives focus on modeling a consistent problem (*i.e.,* a problem that has at least one solution). Given their limited reusability, multiple instances of these assets are needed to make a successful solution domain translation [2].

3. Motivating Example and Open Issues

In order to introduce the research problems treated along this paper, we present an example to illustrate conflicting configurations related to the domain of Information Technology (IT) investment decisions.

3.1 A Domain on IT Investment Decisions

Figure 2 shows a subset of a feature model that was created to represent most of the decisions considered in a decision-making process used for planning Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM) investments. These enterprise information systems are composed by modules that are implemented and configured in order to fulfill different business needs. However, organizations should have a decision process in order to select the appropriate characteristics that could enhance its operation. These processes involve multiple stakeholders which can be viewed as decision-makers. Given their different expertise and points of view, decisions often vary between them and can result into conflicts. This resolution task demands time and effort to seek an unanimous set of decisions.

The *decisions* branch contains 12 decision types defined as decision categories represented by non-atomic features, and 125 decision options which are selection alternatives to different decision types and are represented as atomic features. The *implications* branch contains 219 consequences of the decision options selection which are defined in terms of business dimensions such as costs, time, and human resources. Decisions and implications features are related through 192 cross-tree constraints. [1]

In this example, we can identify three decision types at the *decisions* branch (*i.e.,* business drivers, business systems with both ERP and CRM features, and solution type). Each of the atomic features (feature leaves) is a decision option related to the given parent feature (*e.g.,* improve customer support, management module, commercial solution). There are nine decision options related to the modeled decision types. In addition, two cross-tree constraints among decision feature options are represented as logical propositions at the bottom center of the presented feature model.

The *implications* feature is parent of six additional non-atomic features: *impact variable* that defines the task or subject affected by the implication (*e.g.,* licensing); *dimension*

[1] The complete representation of the feature model can be found at: http://ticsw.uniandes.edu.co/ITGovernancePublic/FeaturesModel.xml

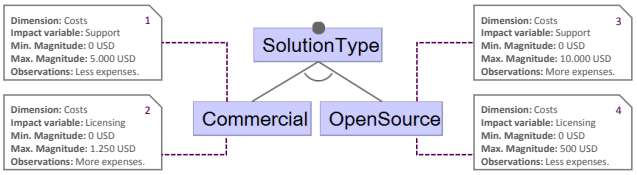

Figure 3: Non-functional properties example.

which represents the business dimension related to the implication (*i.e.,* costs, time, human resources); *unit measure* which defines the unit of the defined magnitude (*e.g.,* USD); *magnitude* that defines a numeric value related to the implication; *frequency* which defines the periodicity on which the implication affects the organization; and *observations* that are semantic additional information for the stakeholders (*i.e.,* it could be seen as important comments for the decision-makers). Figure 3 illustrates features of cost implications as if they had been specified as feature attributes. Implications that are related to decision options through cross-tree constraints are represented as annotations (rectangular boxes) on their corresponding feature (dotted connection line). Each decision option could have multiple implications associated to a unique dimension. For example, if the *commercial* decision option is selected, the organization should pay between 0 and 5.000 USD each month for the system *support*. In contrast, a monthly cost between 0 and 10.000 USD is required for the same activity when selecting the *open source* option.

Conflicting configurations. Figure 4 shows two conflicting configurations performed by different stakeholders on the presented feature model. Each configuration has a list of decision types and decision options that are selected (+ symbol). Their selection supposes the deselection (− symbol) of other features according to the defined constraints.

As it can be seen, for the *business drivers* decision type, stakeholder 1 selected the option *improve customers support*. However, this selection supposes the deselection of the option *reduce delivery time* for the same decision type. The conflict arises when the second stakeholder selects this last feature, forcing the deselection of the *improve customers support* which was previously selected by the first stakeholder. There is an evident conflict at this first decision. At the same time, the selection of the *improve customer support* implies the selection of the *customers CRM* module according to the second cross-tree constraint; while the selection of *reduce delivery time* implies the selection of the two *ERP* modules according to the first cross-tree constraint. *ERP* and *CRM* are exclusive between them so we obtained a second set of conflicts between the stakeholders' decisions. Finally, stakeholder 1 selects a *commercial solution type*, and stakeholder 2 selects an *open source solution type*. Nevertheless, these two options are exclusive between them so a third set of conflicts appeared between the two configurations.

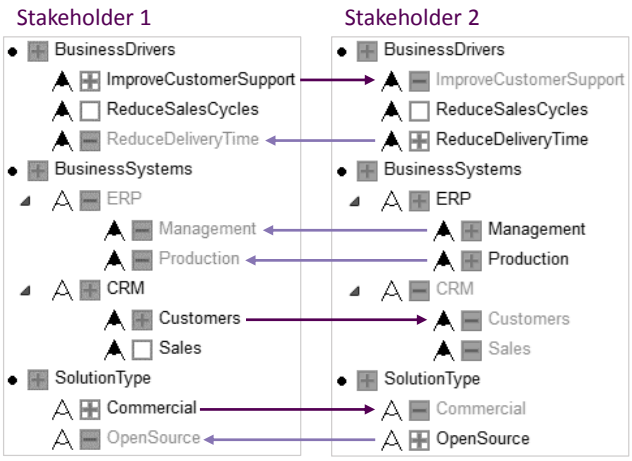

Figure 4: Conflicts among configurations.

3.2 Problems with Conflicts Between Configurations

According to the exposed motivating example as well as to the studied related work, we identify the following two main issues with respect to conflicts between decision products:

Problem 1: Conflicts solving does not take advantage of non-functional properties. Most research projects related to conflicts solving focus on avoiding inconsistencies between configurations, or they proposed alternatives that does not simultaneously take into account both non-functional properties related to each decision option, and stakeholders' decisions. Currently, these approaches provide configuration solutions based on modifying the minimal set of features states between all stakeholders' decisions [22], or on discarding these decisions and generating a new configuration based on a cost function (*cf.* Section 7). Even more, conflicts solving proposals do not take into account the overall business objectives over those non-functional properties such as costs, time, and human resources. Therefore, technology resolution capabilities are not completely aligned with the organization concerns; current approaches can represent propositional relationships but optimization and/or numeric definitions are poorly represented. In our motivating example, implications are not taken into account when defining the best solution for conflicting configurations.

Problem 2: High feature model complexity when representing decisions in a decision-making process. Dimensions such as costs, time, and human resources are associated to IT decisions selection as implications. When representing decisions of a decision-making process in a standard feature model, the organization needs to map associations among the decisions and the implications branches in the feature model. Therefore, the modeler should create the needed features at the implications branch, as well as the required constraints that generate the real mapping. Hence, the created constraint should include the decision option as a condition

and the given implications features as consequences. These features should define dimensions, frequencies, unit measures, magnitudes, impact variables, and other representative implication attributes. Consequently, the introduction or edition of a constraint could suppose a manual error over the model consistency, as well as a non-maintainable task that requires the identification of the desired constraint and its correspondent change [12].

```
1 Attribute_Types {
2 create Costs01: Costs measuredIn "USD" each "Month";
3 create HR01: HumanResources measuredIn "People"
4     each "Month";
5 create Time01: Time measuredIn "Months" each "Year";
6 }
```

Listing 1: Attribute type specification in CoCo.

4. CoCo: A DSL for Decision Rules

This section describes how feature attributes and decision rules are specified through a DSL named CoCo (*Co*nflicts among *Co*nfigurations resolution). Both types of specifications are associated to business dimensions such as costs, time, and human resource, in order to favor separation of concerns and maintainability. At the same time, CoCo seeks to be a compositional language that allows the definition of complex expressions while articulating simpler components.

By its side, feature attributes represent non-functional properties such as the implications described in Section 3.1. Decision rules are sets of optimization and hard-limit functions applied to the total value of business dimensions. This total value is obtained from the sum of the different feature attributes associated values for a given business dimension. In addition, decision rules represent business objectives or restrictions, allowing the alignment of the solution with the organization interests.

4.1 Specification of Non-Functional Properties

The specification of non-functional properties is done with the aim of enriching the feature model with information related to decision options, and to take advantage of these specifications in order to apply decision rules for conflicts resolution within the configuration process. The orchestration of both non-functional properties and decision rules allows the definition of business objectives and the alignment with the organization interests. Non-functional properties are specified through attribute types and feature attributes.

Attribute types are semantic information which define an organization ambit of evaluation, *i.e.,* they are categories that are analyzed during a decision-making process. We define a dimension as a tuple of four values: an *identification* that represents the dimension in both organizational and technical contexts; a dimension such as costs, time, human resources, and risks; a *unit measure*, such as USD in the case of costs dimension; and a *frequency unit* that defines the periodicity of the dimension impact such as each month, quarter, or year, among others. Attribute types could vary from organization to organization, that is why they should be customizable. Although attribute types express the overall ambit of evaluation inside an organization, their related values to atomic features are the real impact that allows project alternatives selection.

Listing 1 provides an example of the specification of attribute types below the keyword *Attribute_Types*. In this example, we present three new attribute types. The first attribute type is identified as *Costs01* and it is specified with a *costs* dimension, with a *USD* measure unit, and with a *monthly* frequency on which implications must be analyzed for related features. The *HR01* attribute type is specified with a *human resources* dimension, with a *people* measure unit, and with a *monthly* frequency. Finally, the *Time01* attribute type is specified with a *time* dimension, with a *month* measure unit, and with a *yearly* frequency.

A *feature attribute* can be defined as an extension of a feature inside a feature diagram. It associates a set of non-functional information that cannot be captured by a feature. A standard feature model (*i.e.,* a model without feature attributes) needs a complex solution in terms of the number and management of constraints and features to represent this data. Hence, in order to centralize the management of these non-functional properties, they are arranged as attributes of atomic features or decision options.

CoCo defines a feature attribute in terms of five well defined properties: an *attribute type reference* that captures the previously defined attribute type identification; an *impact variable* that defines the task or organizational subject, which is directly affected by the non-functional definition; a *lower value* boundary that defines the float minimum value, which could be taken by the feature attribute in a given context; an *upper value* boundary that defines the float maximum value, which could be taken by the feature attribute; and finally, additional *observations*, which have qualitative information that could be relevant for stakeholders.

Listing 2 provides an example of feature attribute specification below the keyword *Feature_Attributes*. In this case, we present the definition of the four feature attributes shown in Figure 3. We present the assignation of two feature attributes to both *commercial* and *open source* decision options. Both options are children of the non-atomic feature *solution type* (*cf.* Figure 2). As it can be noticed, the organization can assign multiple feature attributes to the same feature, and they could be part of different attribute types. If they are part of the same attribute type, the only restriction is that the impact variable should vary in order to avoid inconsistencies or overwriting non-functional properties. Returning to Listing 2, the first non-reserved value of each feature attribute represents the previously defined attribute type identification such as *Costs01*; the second non-reserved value defines an existing feature; the third non-reserved value defines the impact variable such as *support*,

```
Feature_Attributes {
attribute Costs01 onFeature Commercial
     impacts "Support" between 0.0 and 5000.0
     observations "Less expenses.";

attribute Costs01 onFeature OpenSource
     impacts "Support" between 0.0 and 10000.0
     observations "More expenses.";

attribute Costs01 onFeature Commercial
     impacts "Licensing" between 0.0 and 1250.0
     observations "More expenses.";

attribute Costs01 onFeature OpenSource
     impacts "Licensing" between 0.00 and 500.0
     observations "Less expenses.";
}
```

Listing 2: Feature attribute specification in CoCo.

or *licensing*; the next non-reserved values represent in the corresponding order the lower bound value, the upper bound value, and additional observations.

4.2 Specification of Decision Rules

The second objective of CoCo DSL is the specification of decision rules. *Decision rules* are defined by the organization as impartial and conclusive criteria for solving conflicts between stakeholders' decisions without losing their knowledge. They also represent the business objectives and the alignment between our approach and the organization interests. Additionally, these rules complement the propositional restrictions expressed through the feature model constraints, helping to reduce the set of solution alternatives into a subset that focuses on giving an answer to the organization interests. CoCo admits two types of rules: optimization rules, and hard-limit rules. As it can be seen, these rules represent just a subset of the complete business expressiveness. Only optimization and quantitative restrictions can be defined through this DSL first version. By its side, the combination of these rules make of CoCo a compositional language.

Optimization rules define a maximization or minimization task over a dimension; while *hard-limit rules* define a set of boundaries over a dimension, this means that a limit is applied over the sum of the values related to a given attribute type. Hard-limit rules are logical relations over attribute type values that admit the following operators: *less than* (*i.e.,* lt), *less than or equal to* (*i.e.,* leq), *equal to* (*i.e.,* eq), *greater than* (*i.e.,* gt), and *greater than or equal to* (*i.e.,* geq). Simultaneously, the company could define an optimization function that minimizes or maximizes the product value according to the defined hard-limit. Multi-criteria optimization with CoCo is planned for future work.

After the specification block ends, the organization should specify the rules to be executed. In that way, they can refer to the decision rule name or to the keyword *all* for executing all the rules defined in the corresponding decision rules block (*cf.* execution of rules in Section 4.3).

```
Decision_Rules{
//This is an optimization rule
decisionRule optimization R1: minimize Costs01;

//This is a hard-limit rule
decisionRule hardLimit R2: Costs01 leq 15000.00;
}

execute: R1, R2;
```

Listing 3: Decision rules specification in CoCo.

In the motivating example, the company decided to maximize the investment costs while assuring that these costs do not exceed 15.000 USD. Listing 3 illustrates the proposed specification for the previous exposed decision rules. Decision rules should be defined below the label *Decision_Rules*, and they could be both optimization and hard-limit rules. Each optimization rule defines as first non-reserved value the rule identification; it is followed by the desired optimization function represented by the terminal symbols *maximize* or *minimize*; finally, the affected attribute type identification (*e.g., Costs01*, *Time01*, or *HR01*) is specified. By its side, each hard-limit rule defines as first non-reserved value the rule identification, and as second non-reserved value the affected attribute type identification; these values are pursued by a hard-limit logical relation terminal symbol (*i.e., lt*, *leq*, *gt*, *geq*, or *eq*); and the rule ends with a float value related to the defined hard-limit. Finally, the organization specifies the sequential execution of both decision rules *R1* and *R2*, taking advantage of the compositionality of CoCo.

4.3 DSL Syntax

CoCo uses a set of forms and keywords employed during the specification of attribute types, feature attributes, and decision rules. We show the specific forms and the corresponding keywords for each mentioned element.

Attribute types. CoCo uses the keywords *create*, *measuredIn*, and *each* to specify an attribute type. The label *Attribute_Types* is used once to specify the attribute types definition block, which is encapsulated between curly braces. An attribute type is specified as follows:

```
create <attributeTypeId>: <dimensionName>
   measuredIn <unitMeasure> each <frequencyUnit>;
```

Feature attributes. CoCo uses the keywords *attribute*, *onFeature*, *impacts*, *between*, *and*, and *observations* to specify feature attributes. The label *Feature_Attribute* is used once to define the feature attributes block. A feature attribute is specified as follows:

```
attribute <attributeTypeId> onFeature <featureName>
  impacts <impactVariable> between <minimumValue>
  and <maximumValue> observations <observations>;
```

Decision rules. CoCo encapsulates decision rules, which are specified with the keyword *decisionRule*, by using the label *Decision_Rules*. For optimization rules CoCo provides the keywords *optimization*, *maximize*, and *minimize*; and for hard-limit rules it is used *hardLimit*, and *leq, lt, eq, geq*, or *gt*. A decision rule is specified as follows:

```
decisionRule optimization <ruleId>:
  (maximize $|$ minimize) <attributeTypeId>;
decisionRule hardLimit <ruleId>: <attributeTypeId>
  (leq | lt | eq | geq | gt) <boundaryValue>;
```

The organization can specify multiple decision rules related to both optimization and hard-limit categories. The execution of the rules is defined out of the decision rules block after the keyword *execute*. Therefore, CoCo proposes four different ways of defining the rules execution:

```
execute : <optimizationRuleId>;
execute : <optimizationRuleId>, <hardLimitRuleId>;
execute : <hardLimitRuleId>, <optimizationRuleId>;
execute : all;
```

The first expression defines the execution of just one optimization rule by using its identification. The next two expressions specify the execution order between two rules of different type. Finally, the fourth expression executes all the rules define under the *Decision_Rules* label. If this block has more than one rule of each type, the interpreter takes the first specified rule of each type and executes them.

4.4 DSL Advantages

CoCo uses a natural language to specify non-functional properties and decision rules at a high level of abstraction but also with the capability of transforming these specifications into a CSP. CoCo's expressiveness is aligned with real context business needs in which both technical and non-technical stakeholders could define a set of decision rules that conform to the organization interests.

By its side, CoCo's compositionality can be seen through its completely alignment to a feature model that represents decision options and categories in a complete decision process, and to the possibility of integrating simpler expression into a complex business objective. At the same time, CoCo complements the propositional restrictions defined through the feature model constraints, with optimization and hard-limit restrictions that can no be represented through the variability model. In order to accomplish these goals, CoCo relies on developed technologies extending their capabilities. The XML schema of FeatureIDE [20], a modelling Eclipse plugin which was used to represent the standard feature model, was adapted to allow extended feature models. In that way, CoCo enhances functionalities of real tools with the objective of favoring the business decision process characterization and the conflicts among configurations resolution.

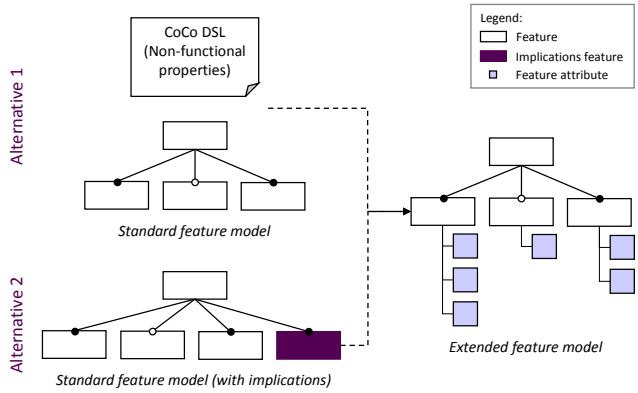

Figure 5: Feature model extension with non-functional properties.

5. A Model Transformation Chain for Conflict Resolution

This section describes the MTC implemented to transform the strategy inputs into a CSP that can be executed in order to solve conflicts among configurations. The strategy is designed to extend the DSL compositionality, where each input of the MTC allows the creation of a CSP representation which encodes the semantics of all elements.

The implementation of the strategy considers two alternatives with a unique resolution path. The first alternative uses as inputs a standard feature model and non-functional properties specified with CoCo. The second alternative was created due to the design of the feature model presented in Section 3.1. This alternative considers a standard feature model that contains non-functional properties (*i.e.,* implications) modeled as features, and relationships between these properties are modeled as constraints.

The MTC will be described in two parts: the extension of the feature model with non-functional properties, and the creation of decision rules that will be part of the conflicts among configurations solving strategy. This MTC is partially developed using tools as: FeatureIDE for managing feature models in a XML format [20]; Xtext for the CoCo DSL specification [23]; Epsilon languages such as Epsilon Generation Language (EGL) for model to text transformations, and the Epsilon Object Language (EOL) used for the model to model transformations [13]; and the CSP solver Choco [10].

5.1 Feature Models with Non-Functional Properties

A feature model that represents decisions in a decision-making process could be extended with non-functional properties that represent certain business restrictions. Figure 5 illustrates the two alternatives proposed in the MTC for achieving this objective.

The first alternative applies for any standard feature model with a set of defined decision types and options. This alternative receives as inputs the feature model and the CoCo DSL. The *CoCo DSL* generates a *non-functional properties*

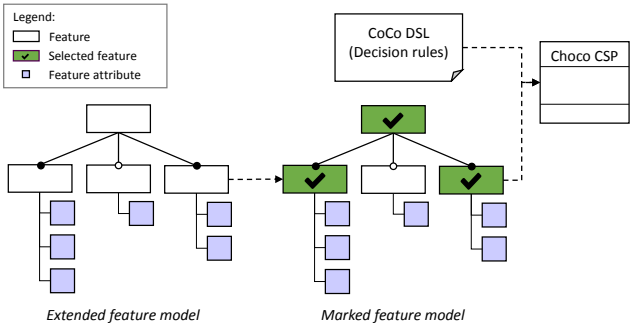

Figure 6: Feature model transformation into a CSP.

model through a text to model transformation (T2M). Then the created model is transformed into an *EOL program* that will be used to extend the original *standard feature model* into an *extended feature model*. This EOL program contains the information related to the non-functional properties. In Figure 5 the feature attributes are represented through the light-purple squares related to their corresponding feature.

The second alternative was strictly developed for our motivating example, but it could be used for any other standard feature model that defines implications as it was described in Section 3.1. In Figure 5 the implications branch is represented through the dark-purple feature. Through this path the *standard feature model (with implications)* is transformed into an *extended feature model*. This is done while taking the implications features and their associated cross-tree constraints, deleting them from the feature model, and converting them into feature attributes related to a specific feature. It is important to notice that the *extended feature model* created at the end of both alternatives has the same XML schema.

5.2 Feature Model Transformation into a CSP

Once the *extended feature model* is obtained (*cf.* Section 5.1), configurations and decision rules specification could be taken into account in order to propose a set of *solution decision products* based on the business objectives. Figure 6 illustrates this final set of tasks related to the proposed MTC.

At this point, stakeholders define their decisions through a configuration process. This will result into a set of *configurations* that could be conflicting among them. All the selected features of all configurations are combined into one configuration. This configuration presents conflicts if it is invalid. Therefore, both the extended feature model and the configurations are inputs to a parser which is responsible of adding the attribute *selected* = *"true"* to the corresponding parent features. The *selected* attribute is created as an attribute over some feature, applying the theory proposed by Karatas et al. [12]. This results into an extended feature model with an additional tag under certain selected features. The green rectangles in Figure 6 represent the selected features by any stakeholder.

```
1 int [] Feature = {x,y};
2 model [i][0] = problem.makeEnumIntVar("Feature",
3 Feature);
4 model [i][1] = problem.makeEnumIntVar("VFeature",
5 0, 1);
6 configs.put("Feature",i);
7
8 problem.post(problem.leq(valuesSum, 15000));
9 problem.minimize(valuesSum, false);
```

Listing 4: Choco code generated from CoCo specifications.

Afterwards, the marked feature model and the specification of *decision rules* through the *CoCo DSL* are used as inputs of a model to text transformation that combine them to generate a CSP model executed by the solver Choco. This transformation results into a Java class with the CSP implementation. When this asset is executed, the organization obtains a set of solution decision products. The mapping between the feature model and the CSP is based on the proposal Benavides et al. [3]. Listing 4 illustrates a segment of the generated Choco specification containing the transformations of feature attributes, constraints, and decision rules.

First, for features and feature attributes we applied the guidelines proposed by Benavides et al. [3] in which *i)* features are set as variables, and *ii)* their domain consists of *true* and *false*. However, the maximum and minimum magnitudes from the dimension implication associated to a feature attribute are used for creating an additional variable *Feature*, which defines the implication magnitude finite domain [9] [21]. The minimum value x is related to the minimum magnitude found on all the feature attributes associated to the selected dimension; the maximum value y is calculated by adding the maximum magnitudes from all the implications related to the feature. The *Feature* variable is then assigned into a matrix tuple, *model*, and depending on if the feature appears on the configuration set, it will have a domain of $\{0, 0\}$ or $\{0, 1\}$. Finally, with the aim of improving performance, each feature location i in the matrix is stored on a hash map structure (*cf.* Lines 1-6 in Listing 4).

Second, feature model constraints are translated to Choco following Benavides guidelines, where relationships among features result into a CSP constraint [3]. Table 1 summarizes the proposed mapping. By this means, *mandatory* relationships are translated into Choco's *and* function where the selection of the implicated features of the relationship is mandatory, thus all children should have a *true* selection state (*i.e.,* feature value equal to 1); *optional* relationships are managed in the same way by using Choco's *or* function, and the selection state for all children under the relationship can be *false* (*i.e.,* feature value equal to 0) or true; the same mapping happens for *or* relationships, where at least one sibling feature should have a value equal to 1; lastly, *alternative* relationships supposed the usage of Choco's *ifOnlyIf* function, where if a child has a *true* selection state its siblings should have a *false* state. *Children selection* and *parent des-*

Table 1: Feature model translation into a CSP problem.

Constraint	Choco function	Feature values
Mandatory	*and*	$f = 1, \forall f \in children\ features$
Optional	*or*	$f = \{0, 1\}, \forall f \in children\ features$
Or	*or*	$f = \{0, 1\}, \forall f \in children\ features$ *with at least one selected sibling feature*
Alternative	*ifOnlyIf*	$f_s = 1\ and\ f_d = 0$, *such that* $f_s \in children\ features, f_d \in (children\ features \setminus f_s)$
Children selection	*implies*	$f_c = 1 \rightarrow f_p = 1$, *such that* f_c *is children of* f_p
Parent deselection	*implies*	$f_p = 0 \rightarrow f_c = 0$, *such that* f_c *is children of* f_p
Requires $(p \rightarrow q)$	*implies*	$p = 1 \rightarrow q = 1$, *for any* $p, q \in features$
Excludes $(p \rightarrow \neg q)$	*implies*	$p = 1 \rightarrow q = 0$, *for any* $p, q \in features$

election constraints are represented through Choco's *implies* function. Each child selection/deselection implies its parent selection/deselection. At the same time, *requires* and *excludes* constraints are also represented through Choco's *implies* function. For the *requires* relationship the selection of the antecedent implies the selection of the consequent, and for the *excludes* relationship the selection of the antecedent implies the deselection of the consequent. Further mapping between feature models and CSP can be found at [3].

Lastly, decision rules for a given attribute type are translated to Choco while using *lt* (less than), *leq* (less than or equal to), *gt* (greater than), *geq* (greater than or equal to), or *eq* (equal to) functions for the hard-limit expression, and *maximize* or *minimize* functions for the optimization segment. Hence, all the related attribute type values of each selected feature are added to a sum variable, that respect and conform to the specified constraint (*cf.* Lines 8-9 in Listing 4). For our decision rules specification example (*cf.* Listing 3), the hard-limit rule is specified in Line 8 of Listing 4, while the optimization rule is defined in Line 9. Afterwards, the Choco CSP is executed obtaining the desired set of configurations that solves the stakeholders' decision conflicts based on the decision rules. Therefore, we obtain a set of products that are conform to business objectives.

6. Assessment

This section illustrates CoCo's compositionality results obtained while using our solution for conflicts resolution over well-defined decision scenarios. It also illustrates the complexity reduction of decision-based feature models when introducing non-functional properties in order to represent business implications.

6.1 Experiment 1: Scenarios for Conflicts Resolution

This experiment is designed to evaluate the effectiveness of the proposed compositionality strategy for solving conflicts among configurations during a decision process. With this objective, we propose a set of decision scenarios where the number of stakeholders and the meaning of decision rules vary. Then, the MTC is executed according to these parameters. All decision scenarios were executed on a machine with an Intel(R) Core(TM) i7-4700MQ processor with 2,40GHz, an 8,00GB RAM, and a Windows 8 64-bit Operating System. The Java Virtual Machine was configured with a maximum memory allocation pool of 3,072GB.

We defined five stakeholders' configurations over our motivating example. The configurations performed by these stakeholders were merged into one invalid product (*cf.* Section 5.2) through four testing sets: the first set used the merged product between configuration 1 and configuration 2; second set employed the merged product between configuration 1, configuration 2 and configuration 3; third and fourth set used a merged configuration that included configuration 4 and 5 in the same way.

In addition, we considered the three attribute types defined in Listing 1, thirty boundary limits related to each attribute type, one maximize optimization rule, one minimize optimization rule, and one hard limit decision rule. Hence, the resulting combination of the four configurations testing sets with these CoCo specifications, resulted in a set of 720 decision scenarios (*e.g.,* for the *first configuration testing set* we apply a maximization rule to the *Costs01* attribute type, and a hard-limit rule with boundary value of *250.000 USD*).

The boundary limits related to the three attribute types were defined based on the sum of the feature model maximum values related to each attribute type, presented in Section 3.1. In particular, *Costs01* attribute type had an initial restriction value of 250.000 USD with and incremental interval of 250.000 USD. Hence, the 30th costs restriction value was 7.500.000 USD. *Time01* attribute type had an initial restriction of four hours as well as an incremental interval of four hours. *HR01* attribute type had an initial restriction and an incremental interval of one person. For each decision scenario, we measured two different aspects: configuration total value related to the specified attribute type that minimizes the restriction, and configuration total value related to the specified attribute type that maximizes the restriction.

The resulting configurations total value related to both *Costs01* and *HumanResources01* attribute types are shown in the logarithmic graph in Figure 7. Discontinuous line zones represent the cases in which there was no solution or the solver was not able to give an answer. To understand the values presented in this figure, we analyze the results highlighted over each chart with a purple circle. The solid outlined circle represents the value related to the configuration that maximizes the restriction, whereas the dotted circle represents the product that minimizes the decision rule. Both elements apply only to the series of 2 configurations (*i.e.,* the first configuration's testing set). Thus, for the *Costs01* attribute type, the suggested decision configuration had a maximum cost of 250.000 USD and a minimum cost of 300 USD; and for the *HR01* attribute type the product that maximizes the total value had a magnitude of three people

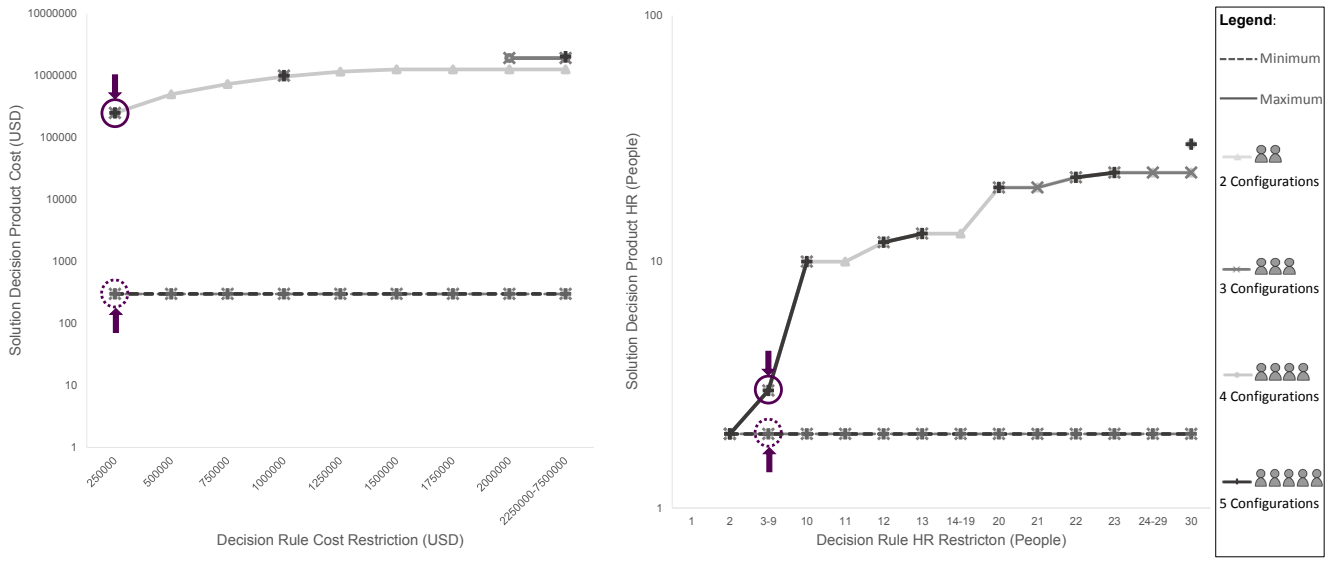

Figure 7: Decision rules over costs and human resources dimensions.

and a product with a minimum magnitude of two people. In this case, the investment decisions fulfill the business restrictions. For the *Time01* attribute type all sets had the same behavior: from 4 to 32 hours did not exist a solution (based on the feature attributes related to each decision); from 36 to 120 hours the maximum and minimum configuration had a value of 36 hours for the four configurations testing sets.

After the decision scenarios execution, we confirmed the generated products validity, *i.e.,* its conformance to the feature model constraints. This result was obtained for all three attribute types. To take advantage of CoCo's compositionality, we combined two different decision rules (one for hard-limit constraints and one for optimization constraints) for each decision scenario what increased the complexity for reaching the business objectives. The resulting products conformed to the defined decision rules and also to the feature model structure. These results validated the goals of the compositionality strategy in which each input kept its meaning and its expressiveness during the resolution path. If there did not exist a product that satisfied the applied decision rule, the solver did not returned a configuration which was the expected behavior. Nonetheless, for certain decision scenarios, the solver threw an exception obtaining no solution at all.

6.2 Experiment 2: Non-Functional Property Specification and Feature Model Complexity

This experiment was executed with the objective of analyzing the feature model complexity while adding non-functional properties to it. In that way, we apply the presented strategy to the motivating example presented in Section 3.1. For our motivating example, we followed the second alternative path (*cf.* Section 5). Table 2 illustrates the differences among the input standard feature model and the output extended feature model. The extended solution had a 61,75% reduction of features, a 63,66% reduction on decision options, and 93,23% reduction on cross-tree constraints. Meanwhile, there were created 178 features attributes with their 23 corresponding attributes types. As result, if we add the number of features, cross-tree constraints, feature attributes, and attribute types for each model, we obtain the number of objects to be maintained. In our case, we can see a reduction of 61,69% objects to be maintained when we apply our strategy to the standard feature model. This decrease is likely to have a positive impact in the operation costs.

Table 2: Differences among baseline and extended feature models.

Feature model elements	Baseline model	Generated model	Delta
Features	366	140	61,75%
Decision options	344	125	63,66%
Cross-tree constraints	192	13	93,23%
Feature attributes	0	178	NA
Attribute types	0	23	NA
Maintained objects	**924**	**354**	**61,69%**

According to the obtained results, the strategy implementation allowed the introduction of feature attributes over a previous defined standard feature model. In that way, decision options were enriched with non-functional properties that are the base for using decision rules in a conflict resolution path. At the same time, when reducing the number of features and the cross-tree constraints that relate them, the model reduces its size favoring the maintenance of its components. Particularly, features and cross-tree constraints maintenance is favored while all implication are transformed to feature attributes. The introduction or deletion of feature attributes is simpler and decoupled avoiding the modification of both cross-tree constraints and features.

7. Related Work

There are several approaches focusing on creating conflict avoidance strategies as well as conflicts resolution approaches. For the conflict avoidance strategies, simultaneous or non-sequential configurations are not allowed and that is the differential point from our proposal.

On the conflict avoidance scenario, Mendonça et al. [16] developed a domain-specific propagation algorithm for feature diagrams that provides unit propagation and forward checking. The propagation is done each time the modeler defines the selection state for a particular feature, which result into a non-conflicting configuration. In a similar scenario, Chavarriaga et al. allow the configuration of decision products over multiple feature models [6]. In that way, each stakeholder decides over its own domain or model, though their decisions impact other existing models. A Feature-Solution Graph (FSG) is used to relate features among models by means of *prohibits* and *forces* relationships. Simultaneously, there exists an algorithm that supports conflict explanation through the decision propagation process.

The last research related to this first approach is the one presented by Czarnecki et al. [8]. They proposed the application of staged configurations, which are implemented through feature model specialization or multi-level configuration. In this proposal roles and decision responsibilities are delegated to each stakeholder. Each stakeholder decision could impact others configurations.

On the conflict resolution scenario, Nöhrer et al. propose four strategies for tolerating inconsistencies during decision-making process, while using a SAT solver [17]. By this means, the introduction of inconsistencies over decision configurations are allowed without propagating the selected choices. Nevertheless, they propose a manual resolution based on SAT strategies where users fix the identified conflicts, which supposes a time consuming task where conflicts are automatically detected but conflicts resolution needs stakeholders' intervention. Moreover, White et al. present a constraint-based diagnostic approach that focuses on solving invalid configuration conflicts by minimizing the set of selected features and deselected features from the flawed configuration resulting from stakeholders' parallel decisions. This work is known as Configuration Understanding and Remedy (CURE) which is implemented in four steps: feature model translation into a CSP, conflicting features identification, suggestions of features selection and/or deselection, and valid product generation [22]. However, the resolution proposal is not based on non-functional properties that represent the overall business interests.

Another conflict resolution scenario has been presented by Benavides et al. [5]. They present a framework named FAMA that combines multiple solvers (*e.g.,* SAT, CSP, BDD) on runtime to improve the performance for analyzing standard and extended feature models. However, its API does not include the identification of inconsistencies and conflicts among configurations over extended feature models, and the resolution strategy should be implemented at a low level of abstraction.

Finally, on the inconsistencies tolerant context, Machado et al. proposed SPLConfig, an Eclipse plugin that uses FeatureIDE capabilities in order to derive an optimal product based on customer budget and other non-functional requirements. In that way, they extend feature models with non-functional requirements, and using Search-Based Software Engineering (SBSE) they propose an automatic product configuration method [15].

These conflict-tolerant approaches do not consider a conflicts resolution based on the complete set of stakeholders' configurations, which ignores parts of specialized knowledge. Furthermore, they are missing the definition business-oriented criteria and guidelines during the conflicts resolution path.

8. Conclusion and Future Work

We propose a strategy to support the decision process on an organization with independent stakeholders' configurations, while considering non-functional properties related to certain decision options. Both feature attributes and decision rules were defined as specifications of CoCo DSL, that represent business objectives or restrictions. For this objective, we use extended feature models and decision rules. Extended feature models enrich features semantic by adding attributes that improve their representation. Furthermore, the introduction of decision rules established the guidelines for conflict resolution between stakeholders' decision configurations. The combination of decision rules for excluding some solutions with optimization and hard-limit criteria facilitates the search for an optimal solution that fulfills business needs. We implement our approach through a transformation of the extended feature model, the decision rules, and the stakeholders' configurations into a CSP.

The strategy was evaluated through two different experiments. The first experiment was centered on evaluating the compositionality strategy and the overall results obtained while using our solution for conflicts resolution over a set of decision scenarios. After the decision scenarios execution, we confirmed the generated products validity. Moreover, compositionality goals were accomplished since the resulting products conformed to the specified decision rules and also to the feature model structure. If no solution exists, the solver does not returned a configuration which is the expected behavior. The second experiment was designed with the aim of analyzing feature model complexity after adding non-functional properties as feature attributes to the variability model. In that way, when reducing the number of features and the cross-tree constraints that relate them, the model reduces its size favoring the maintenance of its components and therefore simplifying the feature model complexity.

For future work, we plan to test the performance and scalability of the proposed strategy. Also, an exploration of other types of solvers and algorithms that can manage large problems is desired for improving the implementation. The introduction of new decision rules profiled by means of attribute types is useful to increase the business rules' expressiveness. Finally, new and more complex constraints are required to define and combine decision rules for solving conflicts by considering an impartial and determinant group of constraints.

Acknowledgments

The authors would like to thank Jaime Chavarriaga, Kelly Garcés, and Juliana Alves Pereira for their useful feedback.

References

[1] C. E. Alvarez Divo. Automated reasoning on feature models via constraint programming. Master's thesis, Uppsala University, Department of Information Technology, 2011.

[2] K. Apt. *Principles of Constraint Programming*. Cambridge University Press, New York, NY, USA, 2003. ISBN 0521825830.

[3] D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated reasoning on feature models. In O. Pastor and J. Falcão e Cunha, editors, *Advanced Information Systems Engineering*, volume 3520 of *LNCS*, pages 491–503. Springer, 2005. .

[4] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. Using java csp solvers in the automated analyses of feature models. In R. Lämmel, J. Saraiva, and J. Visser, editors, *Generative and Transformational Techniques in Software Engineering*, volume 4143 of *LNCS*, pages 399–408. Springer, 2006. .

[5] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-cortés. Fama: Tooling a framework for the automated analysis of feature models. In *First International Workshop on Variability Modelling of Software Intensive Systems (VAMOS)*, pages 129–134, 2007.

[6] J. Chavarriaga, C. Noguera, R. Casallas, and V. Jonckers. Propagating decisions to detect and explain conflicts in a multi-step configuration process. In J. Dingel, W. Schulte, I. Ramos, S. Abrahão, and E. Insfran, editors, *Model-Driven Engineering Languages and Systems*, volume 8767 of *LNCS*, pages 337–352. Springer International Publishing, 2014. .

[7] K. Czarnecki and U. W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000. ISBN 0-201-30977-7.

[8] K. Czarnecki, S. Helsen, and U. W. Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.

[9] Á. Horváth and D. Varró. Dynamic constraint satisfaction problems over models. *Software & Systems Modeling*, 11(3): 385–408, 2012.

[10] N. Jussien, G. Rochart, and X. Lorca. Choco: an Open Source Java Constraint Programming Library. In *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming*, pages 1–10, Paris, France, 2008.

[11] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[12] A. S. Karataş, H. Oğuztüzün, and A. Doğru. From extended feature models to constraint logic programming. *Science of Computer Programming*, 78(12):2295 – 2312, 2013.

[13] D. Kolovos, L. Rose, and R. Paige. The Epsilon Book. `http://www.eclipse.org/epsilon/doc/book/`, January 2013.

[14] K. Lee, K. Kang, and J. Lee. Concepts and guidelines of feature modeling for product line software engineering. In C. Gacek, editor, *Software Reuse: Methods, Techniques, and Tools*, volume 2319 of *LNCS*, pages 62–77. Springer, 2002. .

[15] L. Machado, J. Pereira, L. Garcia, and E. Figueiredo. SPLConfig: Product configuration in software product line. In *Brazilian Conference on Software (CBSoft), Tools Session*, pages 85–92, Maceio, Brazil, September 2014.

[16] M. Mendonça. Efficient reasoning techniques for large scale feature models. Master's thesis, University of Waterloo, Waterloo, 01/2009 2009.

[17] A. Nöhrer, A. Biere, and A. Egyed. A comparison of strategies for tolerating inconsistencies during decision-making. In *16th International Software Product Line Conference, Volume 1*, pages 11–20. ACM, September 2012. .

[18] P. Schobbens, P. Heymans, and J.-C. Trigaux. Feature diagrams: A survey and a formal semantics. In *Requirements Engineering, 14th IEEE International Conference*, pages 139–148, Sept 2006.

[19] S. Soltani, M. Asadi, D. Gašević, M. Hatala, and E. Bagheri. Automated planning for feature model configuration based on functional and non-functional requirements. In *16th International Software Product Line Conference - Volume 1*, pages 56–65. ACM, 2012. .

[20] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich. FeatureIDE: An extensible framework for feature-oriented software development. *Sci. Comput. Program.*, 79: 70–85, 2014.

[21] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and M. Toro. Explanations for agile feature models. In *1st International Workshop on Agile Product Line Engineering (APLE'06)*, 2006.

[22] J. White, D. Benavides, D. Schmidt, P. Trinidad, B. Dougherty, and A. Ruiz-Cortes. Automated diagnosis of feature model configurations. *Journal of Systems and Software*, 83(7):1094 – 1107, 2010. SPLC 2008.

[23] Xtext. Xtext documentation. `http://www.eclipse.org/Xtext/`, 2014.

[24] A. Yie, R. Casallas, D. Deridder, and D. Wagelaar. Realizing model transformation chain interoperability. *Software & Systems Modeling*, 11(1):55–75, 2012.