Handling Regulatory Goal Model Families as Software Product Lines

Anthony Palmieri[1], Philippe Collet[1(✉)], and Daniel Amyot[2]

[1] Université Nice – Sophia Antipolis CNRS, I3S, UMR 7271, 06900 Sophia Antipolis, France
palmieri.anthony@etu.unice.fr, philippe.collet@unice.fr
[2] School of EECS, University of Ottawa, Ottawa, Canada
damyot@eecs.uottawa.ca

Abstract. Goal models can capture the essence of legal and regulation statements and many of their relationships, enabling compliance analysis. However, current goal modeling approaches do not scale well when handling large regulations with many variable parts that depend on different aspects of regulated organizations. In this paper, we propose a tool-supported approach that integrates the Goal-oriented Requirement Language and feature modeling to handle regulatory goal model families. We show how they can be organized as a Software Product Line (SPL), ensuring the consistency of the SPL as a whole, and providing an adapted derivation process associated to a feature model configuration. The proposed approach is also evaluated on large generated SPLs with results suggesting its capability to address scalability concerns.

Keywords: Goal modeling · Goal-oriented requirement language · Legal compliance · Variability · Software product line

1 Introduction

Goal-oriented modeling has been successfully applied to capture and reason about many forms of requirements, being functional, non-functional or even legal [1,2]. In the regulatory domain, many requirements have been modeled and used by regulators for compliance and performance monitoring across regulated organizations. Such models are crucial to improve the suitability, fairness and correctness of evolving regulations and laws. In many contexts, goal-oriented regulation models may be very large, with models containing more than one thousand elements [3,4]. Moreover, it is often the case that regulations are distinguished according to organization types (e.g., reflecting their location, their size, etc.), or simply to different interpretations or analyses [5].

In [5], an approach is proposed to capture a generic goal model with ITU-T's *Goal-oriented Requirement Language* (GRL) [6,7] as a family, enabling one to extract individual members of the family (e.g., a goal model targeting a specific type of organization) for compliance analysis. However, this approach only

J. Zdravkovic et al. (Eds.): CAiSE 2015, LNCS 9097, pp. 181–196, 2015.
DOI: 10.1007/978-3-319-19069-3_12

supports one dimension of variability, e.g., the type of organization, at a time. In large regulatory requirements, the different family members are characterised by complex configurations that expose a high degree of variability. This situation is representative of what is tackled by the *Software Product Line* (SPL) paradigm [8].

In this paper, we propose a tool-supported approach, based on GRL and SPL techniques with feature modeling, to handle regulatory goal model families on a large scale, with many variability dimensions. We show how regulatory GRL model families can be organized as a consistent SPL, providing an adapted derivation process. This enables regulators to obtain a valid and tailored regulatory model that fits their context and expectations.

In the following, background and motivation on goal modeling for regulations and the SPL paradigm are described in section 2. In section 3, we give an overview of our proposed SPL for regulatory goal model families. Consistency of the SPL and of its associated derivation process are described respectively in sections 4 and 5. We report on the evaluation of our current prototype in section 6. Related work is discussed in section 7, while section 8 presents our conclusions.

2 Background and Motivation

2.1 Regulatory Goal Model Families

Regulatory contexts are numerous and most existing goal-oriented languages have already been used for legal compliance modeling and assessment [1]. Regulatory goal models are becoming crucial as regulators use them to be more efficient and have standardized viewpoints on the assessed compliance. This is especially the case when evolving from a prescriptive regulation approach, which imposes specific compliance means, to an intention-centric one, which focuses on regulation goals and performance indicators rather than on implementation means [3].

In this work, we use the graphical language GRL [6] to represent goals for several reasons. First GRL provides ways to model and reason about goals and non-functional requirements in a social context. Its standardization within the User Requirements Notation [7], its support for Key Performance Indicators (KPIs), its support of standard annotations (metadata) for domain profiling, its integration of evaluation strategies, and the availability of a mature modeling and analysis tool (jUCMNav [9]) are relevant advantages over many other goal notations. GRL has also been already used with success to measure compliance of business processes with regulations [1], to analyze organizational security requirements [2] and to reason about large outcome-based regulatory requirements [3]. Tool support for importing regulations and handling large legal models is also available [4]. Finally, GRL is also the basis for the first definition of goal model families [5].

Goal model families have been proposed to handle large and complex regulation models that have to be applied to multiple types of organizations. As

separate models would hinder evolution, increase maintenance costs and the risk of errors, a form of generic model must be maintained. In [5], the authors take the hypothesis that a goal language, such as GRL, can be tailored sufficiently to support the concept of a model family. Applying it to a realistic aerodrome security regulations use case, they manage to annotate/tag goal model elements with the type of aerodrome to form a kind of generic model. Being configured, one can obtain a valid GRL model conform to expectations for a specific type of aerodrome. Tailored tool-supported analysis algorithms recompute contributions levels between intentional elements when elements are removed and ensure the consistency of the obtained goal model. The main problem of this approach is that it only handles one set of tags, i.e. aerodrome types in [5], at a time. If several sets of tags have to be managed to describe the variability of a larger and more complex legal context, there is no way to ensure the consistency of either the constraints between these tags or the resulting goal model family. The approach also suffers from usability issues in that situation. We analyse this as a variability management problem as, now classically, handled by the SPL paradigm [8]. Despite different approaches that couple the SPL paradigm to goal modeling (see section 7), to the best of our knowledge, none of them tackles the problem of handling a highly variable goal model family. Most approaches actually use a goal model to guide selection of a particular SPL configuration, e.g., by taking into consideration quality aspects and potential trade-offs [10,11].

2.2 Software Product Lines and Feature Modeling

Promoting systematic reuse of sofware artifacts, SPL engineering aims to manage and generate software variants tailored to the needs of particular customers [8]. This paradigm is now gaining increasing attention in different application domains to efficiently handle software families. It mainly exploits what variants have in common and manages what varies among them. *Feature models* (FMs) are a widely used formalism to model this variability in terms of mandatory, optional and exclusive features organized in a rooted hierarchy, together with propositional *cross-tree* constraints over the features [12]. A example of FM is depicted on the right part of Fig. 1, with *snow_risk* being an optional feature, *type1* to *type3* being mutually exclusive, and some constraints expressing feature implication.

The semantics of a FM [12] characterizes the valid combinations of its features as a configuration. Automated reasoning operations can also be applied on the FM [13], notably as an FM can be encoded as a propositional formula defined over a set of Boolean variables, where each variable corresponds to a feature [14]. A configuration defines one product of the SPL on a conceptual level. It can be used as input to a variability realization mechanism, which processes all relevant realization assets to derive, i.e. create, a concrete software system as variant of the SPL. Annotative mechanisms define a so-called *150% model* that encompasses all possible variations of assets for the entire SPL [15,16]. During derivation, parts of the 150% model that are not relevant to a particular variant are removed, and mechanisms must be provided in the SPL to ensure consistency. On the other hand, compositional variability realization mechanisms

assemble variants by combining elements of assets with a common core to obtain a product [17].

3 A SPL of Regulatory GRL Models

In order to handle regulatory goal model families on a large scale, we propose to organize them following the SPL paradigm. In this section, we discuss and illustrate the proposed SPL organization.

3.1 Rationale and Overview

In order to better manage the different domain elements that can influence the form of a regulatory goal model within a family, we first choose to use a separate feature model (FM) to represent the variability. For the realization mechanism, we propose to follow the approach of Czarnecki et al. [15]. The family contains all regulation possibilities in a so-called 150% model that is represented directly in GRL. We then rely on a *negative* variability principle so that the selection of a feature (within a configuration) will remove from this 150% model elements irrelevant to the configuration, so to obtain a completely tailored product (GRL model) at the end. Consequently, the FM will allow one to configure the 150% model. Together, they make up a regulatory GRL model family.

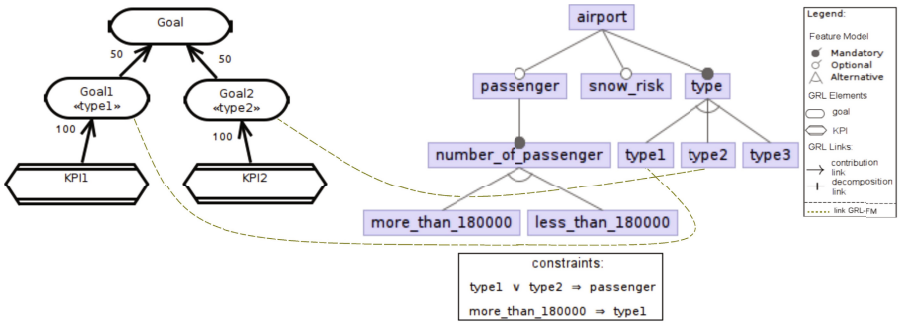

Fig. 1. Architecture of the regulatory goal model SPL

Figure 1 shows how a goal SPL is architected. The 150% GRL model is standard GRL. It contains intentional elements, like goals (e.g., Goal and Goal1), contribution links (→) connecting two intentional elements with quantitative weights ([-100, 100]), AND/OR decomposition links (⁺) connecting elements with sub-elements, and finally KPIs, which convert values from the real world (e.g., KPI1) to a GRL satisfaction level according to a defined conversion method (involving target, threshold, and worst-case values)[1]. Intentional elements can optionally be allocated to actors, not shown here.

[1] In GRL, KPIs can be linked with intentional elements only if they are the source of a contribution or decomposition link.

To relate the FM and the 150% GRL model so that model elements irrelevant in a specific context can be pruned, we use annotations on GRL elements. Since regulatory models are mainly made of goals and KPIs and since KPIs are not meaningful by themselves, we choose to only annotate goals in the following examples (without loss of generality). Goal annotations are made of feature names within the associated FM. To be more expressive in annotations, we consider that they form a propositional logic formula in which terms are feature names, e.g., the goal *Goal1* is annotated with the feature *type1*. During the derivation process, a goal element containing a formula will be in the final product if and only if its associated formula is evaluated to true. Otherwise it will be removed.

Building different goal model SPLs with our approach, we realized that the complexity of annotations should not be too important so to ease their creation and maintenance. A general rule is to use simple formula in the annotations (e.g., and/or relationships between two features) and to move to the feature model more complex constraints. This simplifies annotations, but also enables one to check the consistency of constraints with feature modeling analysis operations [13].

3.2 Consistency Issues

At derivation time, annotations can be valuated with a given valid and final configuration. The resulting formula is evaluated by considering that all selected features are replaced with true, whereas all deselected features are replaced with false. When goals have been removed, the consistency of the GRL model must be preserved so that, to exist in the final product, an intentional element must be linked to the root or have at least an ancestor linked to it.

With such expressiveness in the annotation language, some annotations may be inconsistent. For example an annotation formula could always be true, so that it has not impact on the 150% GRL model. Conversely, a formula which is always false is inconsistent because the associated element can never be in the derived product. More complex issues may arise when the annotation formula, combined with the FM formula, leads to an always-true or always-false result.

Moreover, an annotation has an evaluation context, because an element can be present if its parent (through GRL links) is present as well, i.e., if the parent element is annotated, its formula should be evaluated to true. Consequently we must verify that for each ancestor, all annotations are always consistent.

Finally, another problem is related to the pruning of the 150% model. Removing elements has obviously an impact on the consistency of the GRL model, especially for contribution links. In the considered regulation models, for a given element, the sum of all incoming quantitative contributions should be less or equal to 100. If one removes a linked element contributing to a goal satisfaction, the semantics will be affected as this goal will not be as satisfiable as before.

3.3 Illustrative Example

In this paper, we use as a representative example the security regulations of Canadian airports[2] which was also used in previous work on regulatory goal modeling [3,5]. An extract of the considered GRL model is depicted in Fig. 2, with a more complete one used in section 5. This model illustrates the main elements structuring a regulatory GRL model: goals as intentional elements (e.g., Perimeter Security), contribution links (→) (e.g., the goals Rules regarding signs and Perimeter Security are linked by a quantitative contribution with a weight of 10), and decomposition links (e.g., Perimeter Security and others *AND*-decomposing Airport Compliance). In addition, KPIs are, for example, used for the number of fences that do not comply with *Fence rule2*.

In our example, the airport compliance to regulation rules depends at least on two high-level requirements: the security perimeter and the firefighting equipment/personal. Each such requirement is decomposed into operational and control rules (e.g., rules regarding fences). Furthermore, each rule receives contributions from KPIs measuring the rule compliance level.

However, some requirements directly depend on airport characteristics. For instance airports can be divided into three types (*type1*, *type2* or *type3*) and some requirements must be hidden or shown according to this characteristic. Following our approach, this notion of variability is organised with others (snow risk, influx of passengers) into a feature model, which is actually the FM shown in Fig. 1 for our airport example. If different versions of the goal model were created for different combinations of types of organizations, their maintenance would be cumbersome and error-prone.

In our approach, several goals are thus going to be annotated. For example, the goal *fence rule2* is annotated with the feature "*type2*", hence this requirement will be hidden if the airport is not of type *type2*. As shown on Fig. 2, each requirement that depends on specific characteristics is then annotated. On the "*Rules regarding access control systems*" goal, one can observe another kind of annotation with a combination of features. There is then a dependency between this goal and "*Access control system rule2*" that could result in inconsistencies between annotations. To ensure safe SPL management, all annotations must be well-formed and there should not exist any conflict between them (see section 4).

Moreover, at configuration time, when features are selected (e.g., *type1*) the FM supporting tool enforces the feature model semantics so that features *type2* and *type3* are automatically deselected. After this selection, if we focus for example on "*Rules regarding fences*", elements will be pruned and this rule will only receive a contribution from "*Fence rule3*". Consequently, if we directly apply the GRL algorithm to compute satisfaction, even if all the fences are compliant, the highest reachable satisfaction level would be 33 instead of 99. The associated SPL derivation process should also ensure consistency of contributions in each derived GRL model (see section 5).

[2] http://bit.ly/1w7jwvo

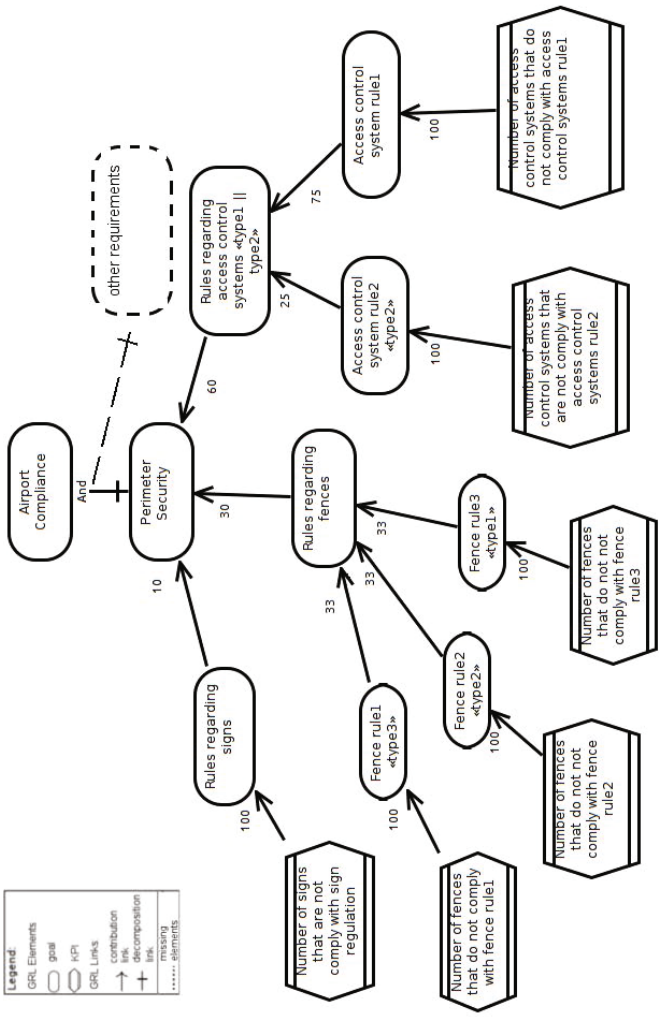

Fig. 2. Extract of the airport regulatory model with annotations

4 Consistency of the SPL as a Whole

As we follow an annotative approach, information is added to a GRL model so that it forms the 150% model, central to the proposed SPL architecture. Checking the consistency of the SPL as a whole boils down to checking the consistency of the annotations on model elements.

4.1 Annotation Consistency

Once the annotation process is performed, several consistency properties must be verified. A first straightforward check consists in ensuring that all annotations are formula composed with feature names from the associated feature model. For each annotation to be semantically consistent, we have to check that it has an impact on the SPL (i.e., the annotation really realizes some variability over the GRL model), and that it has no conflict with the other annotations.

The FM acts as the variability model and can be translated to a Boolean formula to be checked for consistency using SAT-based analysis[3] [18]. A FM is inconsistent if its feature constraints prevent any product configurations; it is incorrect if it contains features that do not belong to any valid product (*dead features*). We can also use the same technique to determine whether a selection of features form a valid configuration. In our approach, an annotation a is valid if and only if its associated formula ϕ_a can be evaluated to false with a given configuration and to true with another one. In other words, there must exist a configuration in which the annotated goal is present and another one where it is absent. This property can be ensured by checking that the conjunction of the FM formula ψ_{fm} with the annotation formula ϕ_a has at least one solution (s_1) to be evaluated to true and another one (s_2) to false: $\exists s_1, s_2 | (s_1 = \psi_{fm} \wedge \phi_a) \wedge (s_2 = \neg(\psi_{fm} \wedge \phi_a))$

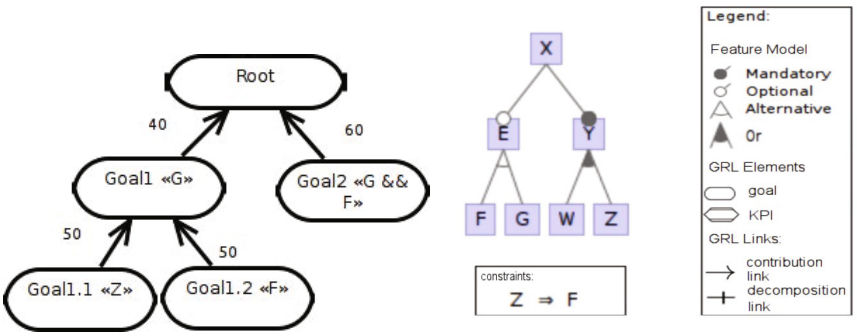

Fig. 3. Multiple inconsistent annotations

[3] Boolean satisfiability is a decision problem about determining whether a Boolean formula evaluates to true for any assignment to its variables.

This verification must be complemented to ensure that two annotations are not in conflict. To do so, we must take into consideration the context of the annotation, i.e., all annotated elements that are ancestors of the element being checked. For example, in Fig. 3, checking consistency of *Goal1.2* is equivalent to checking that after the selection of G, selecting F keeps the configuration valid. Hence, we must check all annotated paths from the root to the leaves. Similarly to single annotation checking, we can check the conjunction of all annotations on a path with the FM formula. But we can also SAT-based check the same property by adding all annotations context as constraints in the feature model. This is equivalent to the previous checking technique, because if one wants that a given path exists in the derived model, one must ensure that, at the end of this path, all annotations are evaluated. The check will then simply be done on the conjunction of the augmented FM formula and the goal annotation formula.

4.2 Illustration

In Fig. 3, several inconsistencies with the feature model and between annotations are illustrated. Note that *Goal1.1* and *Goal1.2*, annotated with "Z" and "F" respectively, are the children of goal element *Goal1*, itself annotated with "G". *Goal2* contains the formula "*G and F*". First, the annotation "*G and F*" is inconsistent because in a configuration, one cannot select G and F due to the affiliation of G and F to the same XOR group in the feature model FM. The conjunction of FM and the formula "*G and F*" will be unsatisfiable.

Second, considering the path composed by elements {*Root, Goal1, Goal1.1*}, the annotation "Z" on *Goal1.1* is not consistent. If we search for a configuration validating all the paths, the context annotation will impose that "G" be selected by the goal "*Goal1*" and that "Z" be selected on the element "*Goal1.1*". But this selection is impossible because if Z is selected, F must be selected too (by the constraint "Z implies F" while F and G belong to an XOR group and cannot be selected together). Finally the path {*Root, Goal1.1, Goal1.2*} is also inconsistent as there exists no configuration where F and G can be selected.

4.3 Algorithm

Applying the rules determined above, we check the consistency of a regulation GRL model as follows: for all paths from the root, we perform a depth-first search (DFS). During this DFS, we build the conjunction of all encountered formula on each traversed path (cf. *getAllFormula* in Algorithm 1). For each different formula obtained, consistency is then checked. Checking the conjunction of each formula on each path is a sufficient condition as it checks that a configuration exists such that the corresponding path can exist and not exist for different solutions. It thus ensures that all elements can be present or removed in the derived GRL model.

Applying this algorithm to Fig. 3, the DFS is performed on the GRL model and creates formulas according to annotations found. ¿From the different paths,

Algorithm 1. SPL consistency checking

```
1. Global variable: grlModel, featureModel
2. function getAllFormula(element, formula, set)
3.     if element is a leaf then
4.         add formula to set                                   ▷ no duplicate
5.     else
6.         for each children w of element do
7.             if w is annotated then
8.                 set ← getAllFormula(w, formula ∧ w.formula, set)
9.             else
10.                set ← getAllFormula(w, formula, set)
11.            end if
12.        end for
13.    end if
14. return set
15. end function
16. function main
17.    getAllFormula(grlModel.getRoot, new formula(), set ← new set())
18.    for each formula in set do
19.        CheckFormulaConsistency(formula ∧ FeatureModel.formula)
20.    end for
21. end function
```

different formulas will be obtained and checked: $G \wedge Z$ is obtained from {Root, Goal1, Goal1.1} and $G \wedge F$ both from {Root, Goal1, Goal1.2} and {Root, Goal2}.

5 Consistent Derivation Process

5.1 Derivation Procedure

Assuming that all annotations are consistent within the SPL, the next step is to ensure that the derivation always gets to a valid product, namely a valid GRL model. Following the negative variability principle, this process (Algorithm 2) will prune goal elements according to a given feature model configuration. We assume that the arguments to call the following algorithm are the root element of GRL model and the *true* logic value to specify that the root element is present.

Algorithm 2. Derivation

```
1. function derive(element, evaluation)
2.     if element is annotated then
3.         evaluation ← evaluateFormula(element.formula)∧ evaluation   ▷ this function evaluates
   the formula on element according to the selection in the FM.
4.     end if
5.     if evaluation is true then
6.         add runtime annotation to keep element   ▷ handles element's shared descendants, if any
7.     end if
8.     if last visit of element then
9.         for children w of element do
10.            derive(w, evaluation)
11.        end for
12.        if ∄ runtime annotations on element then
13.            remove element
14.        end if
15.    end if
16. end function
```

To apply the derivation algorithm, we first consider that a valid and complete configuration has been made on the feature model, as this can be directly ensured by relying on SAT-based analysis [18]. In Fig. 4, we illustrate the potential issues to be handled with a configuration of the airport feature model. We have selected features *type1*, *passenger*, and *more_than_180000*, and unselected *snow_risk*, *type2*, *type3*, and *less_than_180000*.

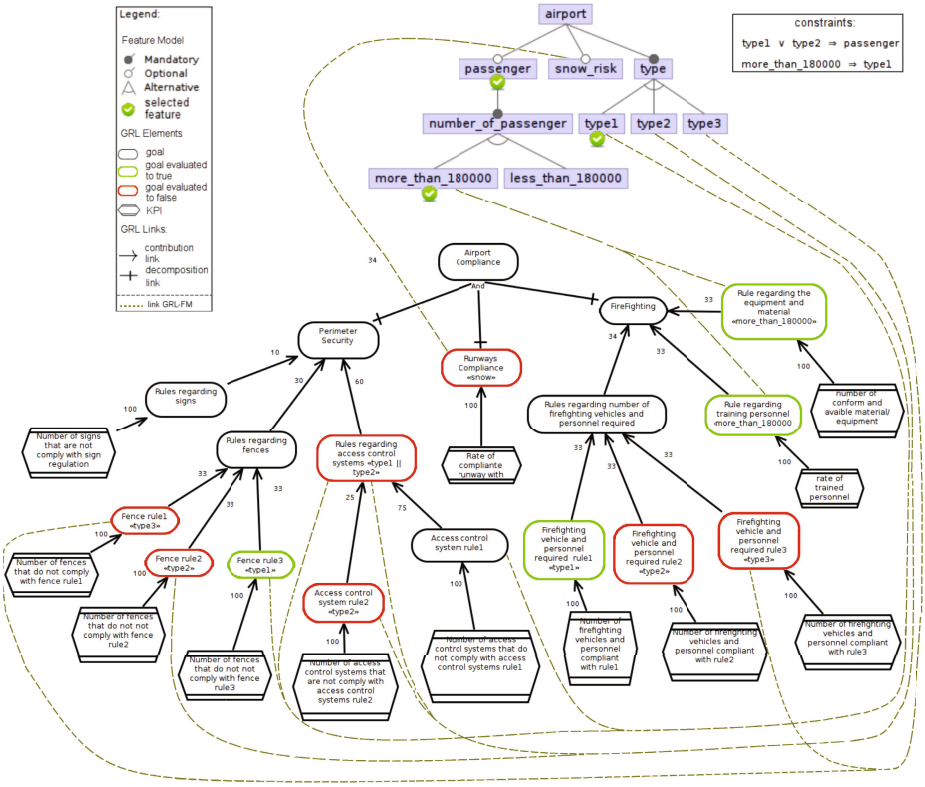

Fig. 4. Airport regulation SPL under configuration (red elements and descendants are pruned, and remaining contributions need to be balanced)

The impact on links must also be studied. Two kinds of links are present in GRL regulations models: decomposition and contribution. For decomposition links, the semantics does not change since the computation of the satisfaction of OR and IOR decompositions is the maximum of the children's values. Similarly, for the AND decomposition, the satisfaction value of the parent will be the minimum value of its children, so if at least one link exists, we can still compute the minimum value. As a result removing an element from the 150% model linked with a decomposition does not impact the semantics of the derived GRL model.

However, there are issues with contribution links. In our derivation example, after pruning, the goal element "*Rules regarding fences*" satisfaction cannot be higher than 33 even if all fences conform to the regulation. The derivation process has thus to be extended to rebalance contributions.

5.2 Maintaining Consistency in Contributions

To deal with the removal of elements linked to contributions, an algorithm has been proposed by Shamsaei in [5]. Assuming that the sum of the contributions cannot exceed 100, it redistributes the weights of removed contribution links among the links of the remaining intentional elements. Consequently, a requirement keeps the same maximum satisfaction level that was reachable before the configuration and pruning. As our context of usage is similar, we can directly reuse Shamsaei's algorithm to obtain consistent contributions by reallocating the lost weights in a proportional manner. For example, in Fig. 4, the removal of goal "*Rules regarding access control systems*", with a contribution of 60, makes the remaining contributions increase: the 10 from "*Rules regarding signs*" becomes 25 (i.e., $10 + (10 \times 60) \div (10 + 30)$) and the 30 from "*Rules regarding fences*" becomes 75 (i.e., $30 + (30 \times 60) \div (10 + 30)$). The sum hence remains the same ($10 + 30 + 60 = 25 + 75$), and the ratios among the remaining links are the same ($10 \div 30 = 25 \div 75$). Similarly, the contribution of "*Fence rule3*" to "*Rules regarding fences*" becomes 99 as this is the only one left ($(33 + (33 \times 66) \div (33)$).

6 Evaluation

6.1 Implementation

The whole approach described in this paper has been implemented. We developed a prototype in Java to support the consistency checking and derivation process of the goal model SPL. Our prototype first relies on the jUCMNav tool [9] and its API for handling GRL models. jUCMNav supports metadata, which are name-value pairs used to annotate any model element. We rely on this functionality to support our SPL formula annotations. As for the feature modeling and reasoning part, our prototype directly uses the FAMILIAR language and Java API [19] to manage feature models and call necessary SAT-based analysis.

The prototype is complemented by a test suite and our airport regulation SPL served as an end-to-end validation of the described algorithms. As our approach aims to manage large and highly variable goal models, we conducted an experiment to observe the behaviour of the consistency checking algorithm on large regulatory SPLs. Our SPL architecture exploits the structure of regulatory GRL models and uses SAT-solving in a way similar to feature modeling [18]. We thus expect good capabilities in handling large regulatory goal families.

6.2 Experimentation

Our experiment consists in measuring the computation time of our consistency checking algorithm on generated goal SPLs. To generate these SPLs, we first use the SPLOT software [20] to create random feature models, and vary both size (i.e., number of features) and cross-tree constraints ratio (CTCR)[4].

On the GRL side, there does not exist any model generator, so we relied on domain expert observations to design and implement a generator of random regulatory models. Due to the complexity and the important number of parameters that should be considered in GRL models, we had to make several design choices. We thus decided that the depth of a regulatory GRL model was not relevant to our study, as the complexity in annotations is due to the number of different sets of formula conjunctions that result from the paths getting from leaves to the root. From previously existing GRL regulation models, we observed that the largest one was composed of 3000 intentional elements, and that the proportion of annotated elements generally does not exceed 20%. In these annotations, 50% of formulas are identical and a formula contains around 2 features on average. We thus randomly generated GRL models with 500 to 5000 elements, with 10% to 30% of annotated elements, and with annotations being 50% to 85% different.

As for the FM characteristics, annotations were made with two kinds of feature models: the first one contains 50 features while the other one has 150 features, which was our upper bound for the largest annotated GRL model. As the CTCR is strongly influencing the complexity of the SAT solving, we varied the ratio of constraints from 10% to 30%, this latter value being a maximum observed on real feature models [18]. Finally, we took randomized feature models of the two kinds (50 and 150 features) that were related to more than 1200 GRL models that have been annotated in several ways.

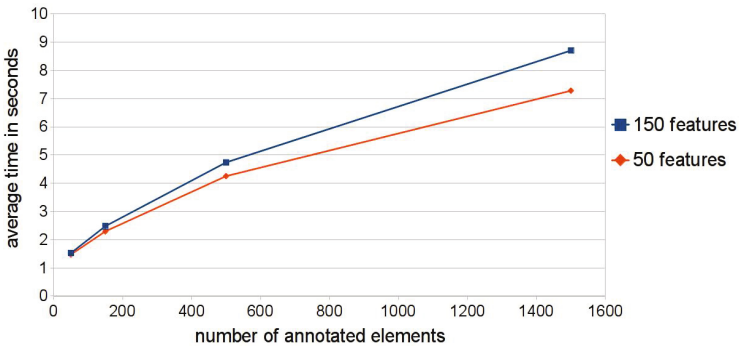

Fig. 5. Consistency checking times (averaged)

[4] The CTCR measures the degree of involvement of features in the constraints, by computing the ratio of the number of features in the cross-tree constraints to the total number of features.

The experiments were run on a dual-core virtual machine with 4GB RAM, representing a laptop on which the creation and the derivation of the goal models SPL could be done in practice. The average times to check consistency of the resulting SPL are shown in Fig. 5. On large SPLs, the average checking time taken is between 4 and 8 seconds. In the worst case, the consistency checking has taken 30 seconds on a GRL model with 5000 elements, including 1500 elements annotated with 85% of different annotations. We were also able to observe that the size of the feature model had little impact on the checking time (around 1 second difference for the more complex GRL models in Fig. 5). Resulting times can be seen as reasonable, as the machine used is not very powerful and the checking is only performed when new annotations or goals are modified.

In addition, we ran similar experiments on the derivation process. Results show that this operation runs in polynomial time as a function of the number of GRL model elements. This is consistent with the derivation algorithm (Algorithm 2), as it is essentially a graph traversal with formula computations.

7 Related Work

The work of Shamsaei et al. on GRL goal families [5] provides a first solution to manage variability of regulatory GRL models. As already said in section 2.1, the proposed solution directly tailors the GRL models with a single set of annotations and cannot deal with complex variability settings. In contrast, our solution makes explicit the captured variability in a feature model, provides consistency mechanisms while reusing their contribution balancing algorithm. As for the architecture of our solution, similar results would have been achieved using the Orthogonal Variability Model (OVM) approach [21]. The FM would have been related to an OVM model representing variants of the goal models, leading to a more explicit but also heavier architecture. We see our solution as more lightweight while being scalable.

Variability on goal models has also been studied according to different dimensions. Similarly to Shamsaei et al. [5], Lapouchnian et al. [10] label model elements with boolean tags, so to represent domain variability and extract a goal model. They also extend the i^* notation to support variations in goal models [22], but they do not support the quantitative evaluation of goal models nor an expressive variability model as in our approach.

Goal-oriented modeling has also been used to complement variability modeling in SPL engineering. For example, feature models are derived from goal models [23,24], consistency checking is performed between the two models [25,26], feature models are preconfigured using stakeholder's objectives [11] or quantitative constraints [27]. However, these approaches do not see the goal model as the product being managed within an evolving family.

8 Conclusion

Current goal modeling approaches do not scale well when handling large and highly variable regulations. We have proposed a tool-supported approach that

integrates the Goal-oriented Requirement Language and feature modeling to handle regulatory goal model families. We have shown how they can be organized as an SPL by annotating a goal model with propositional formula related to features in a feature model. We have provided techniques and algorithms to check consistency of the SPL as a whole and ensure the derivation of valid tailored goal models. Our approach has been evaluated on large generated SPLs, and results suggest its capability to address scalability concerns in a practical time.

Threats to validity concern the simulated models. For the feature modeling part, the simulation parameters have been chosen to generate a wide variety of FMs, with some very high ratios of constraints, so that we are confident that this covers a majority of real FMs. For the GRL part, they are randomly but conservatively generated following patterns of already some developed regulatory GRL models [5]. If applicability is not demonstrated, the structure of the GRL models is quite regular and we expect larger and highly variable regulatory requirements to be captured and exploited with the provided approach. We plan to apply it on real regulatory settings to get new insights. We also want to extend the approach to manage variability against all elements within a GRL model, and explore less structured goal models outside the regulation domain.

References

1. Ghanavati, S., Amyot, D., Peyton, L.: A systematic review of goal-oriented requirements management frameworks for business process compliance. In: Requirements Engineering and Law (RELAW) 2011, pp. 25–34. IEEE CS (2011)
2. Shamsaei, A., Amyot, D., Pourshahid, A.: A systematic review of compliance measurement based on goals and indicators. In: Salinesi, C., Pastor, O. (eds.) CAiSE Workshops 2011. LNBIP, vol. 83, pp. 228–237. Springer, Heidelberg (2011)
3. Tawhid, R., Braun, E., Cartwright, N., Alhaj, M., Mussbacher, G., Shamsaei, A., Amyot, D., Behnam, S.A., Richards, G.: Towards outcome-based regulatory compliance in aviation security. In: 20th IEEE International Requirements Engineering Conference (RE), pp. 267–272. IEEE CS (2012)
4. Rashidi-Tabrizi, R., Mussbacher, G., Amyot, D.: Transforming regulations into performance models in the context of reasoning for outcome-based compliance. In: Sixth International RELAW Workshop, pp. 34–43. IEEE CS (2013)
5. Shamsaei, A., Amyot, D., Pourshahid, A., Braun, E., Yu, E., Mussbacher, G., Tawhid, R., Cartwright, N.: An approach to specify and analyze goal model families. In: Haugen, Ø., Reed, R., Gotzhein, R. (eds.) SAM 2012. LNCS, vol. 7744, pp. 34–52. Springer, Heidelberg (2013)
6. Amyot, D., Mussbacher, G.: User Requirements Notation: The first ten years, the next ten years. Journal of Software **6**(5), 747–768 (2011)
7. ITU-T: Recommendation, Z.151 (11/08) - User Requirements Notation (URN)-language definition, Geneva, Switzerland (2008)
8. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag (2005)
9. Mussbacher, G., Amyot, D.: Goal and scenario modeling, analysis, and transformation with jucmnav. In: ICSE-Companion 2009, pp. 431–432. IEEE CS (2009)

10. Lapouchnian, A., Mylopoulos, J.: Modeling domain variability in requirements engineering with contexts. In: Laender, A.H.F., Castano, S., Dayal, U., Casati, F., de Oliveira, J.P.M. (eds.) ER 2009. LNCS, vol. 5829, pp. 115–130. Springer, Heidelberg (2009)
11. Asadi, M., Bagheri, E., Gašević, D., Hatala, M., Mohabbati, B.: Goal-driven software product line engineering. In: Proceedings of SAC 2011, pp. 691–698. ACM (2011)
12. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Generic semantics of feature diagrams. Computer Networks **51**(2), 456–479 (2007)
13. Benavides, D., Segura, S., Cortés, A.R.: Automated analysis of feature models 20 years later: A literature review. Inf. Syst. **35**(6), 615–636 (2010)
14. Czarnecki, K., Wąsowski, A.: Feature diagrams and logics: there and back again. In: Proc. of SPLC 2007, pp. 23–34. IEEE CS (2007)
15. Czarnecki, K., Antkiewicz, M.: Mapping features to models: a template approach based on superimposed variants. In: Glück, R., Lowry, M. (eds.) GPCE 2005. LNCS, vol. 3676, pp. 422–437. Springer, Heidelberg (2005)
16. Kästner, C., Apel, S., Kuhlemann, M.: Granularity in software product lines. In: ICSE 2008, pp. 311–320. ACM, New York (2008)
17. Batory, D., Sarvela, J.N., Rauschmayer, A.: Scaling step-wise refinement. IEEE Trans. Softw. Eng. **30**(6), 355–371 (2004)
18. Mendonca, M., Wąsowski, A., Czarnecki, K.: SAT-based analysis of feature models is easy. In: SPLC 2009, pp. 231–240. Carnegie Mellon University, USA (2009)
19. Acher, M., Collet, P., Lahire, P., France, R.B.: Familiar: A domain-specific language for large scale management of feature models. SCP **78**(6), 657–681 (2013)
20. Mendonca, M., Branco, M., Cowan, D.: SPLOT: software product lines online tools. In: OOPSLA 2009 companion, pp. 761–762. ACM (2009)
21. Metzger, A., Pohl, K., Heymans, P., Schobbens, P.Y., Saval, G.: Disambiguating the documentation of variability in software product lines: a separation of concerns, formalization and automated analysis. In: RE 2007, pp. 243–253 (2007)
22. Lapouchnian, A., Mylopoulos, J.: Capturing contextual variability in i^* models. In: 5th International i^* Workshop, vol. 766, pp. 96–101. CEUR-WS.org (2011)
23. Yu, Y., do Prado Leite, J.C.S., Lapouchnian, A., Mylopoulos, J.: Configuring features with stakeholder goals. In: SAC 2008, pp. 645–649. ACM (2008)
24. Silva, C.T., Borba, C., Castro, J.: A goal oriented approach to identify and configure feature models for software product lines. In: WER 2011 Worskop (2011)
25. Mussbacher, G., Araújo, J., Moreira, A., Amyot, D.: AoURN-based modeling and analysis of software product lines. Software Quality Journal **20**(3–4), 645–687 (2012)
26. Liu, Y., Su, Y., Yin, X., Mussbacher, G.: Combined propagation-based reasoning with goal and feature models. In: MoDRE 2014 workshop, pp. 27–36. IEEE (2014)
27. Than Tun, T., Boucher, Q., Classen, A., Hubaux, A., Heymans, P.: Relating requirements and feature configurations: a systematic approach. In: SPLC 2013, pp. 201–210 (2009)