Goal-Driven Software Product Line Engineering

Mohsen Asadi, Ebrahim Bagheri, Dragan Gasevic and Marek Hatala

ABSTRACT

Feature Models encapsulate functionalities and quality properties of a product family. The employment of feature models for managing variability and commonality of large-scale product families raises an important question: on what basis should the features of a product family be selected for a target software application, which is going to be derived from the product family. Thus, the selection of the most suitable features for a specific application requires the understanding of its stakeholders' intentions and also the relationship between their intentions and the available software features. To address this important issue, we adopt a standard goal-oriented requirements engineering framework, i.e., the *i** framework, for identifying stakeholders' intentions and propose an approach for explicitly mapping and bridging between the features of a product family and the goals and objectives of the stakeholders. We propose a novel approach to automatically preconfigure a given feature model based on the objectives of the target product stakeholders. Also, our approach is able to elucidate the rationale behind the selection of the most important features of a family for a target application.

1. INTRODUCTION

A software product line (SPL) covers the feasible space of all possible software products for a given domain of interest. In other words, it provides the means for capturing the commonalities of all possible products of a given domain and also addresses variability by covering a comprehensive set of dissimilarities between the products. In SPLs, characteristics of a software system mostly including its functionalities are represented by *Features* [7]. Software product lines have two main lifecycles, namely domain engineering and application engineering. The *Domain Engineering* lifecycle is concerned with representing all of the features of a set of similar/related software systems as a Feature Model (Section 2.3). Later, the *Application Engineering* lifecycle involves the elicitation of the needs, requirements and expectations of the stakeholders to develop a suitable final product based on a given feature model. As the target domain becomes more complex, the structure of the feature model grows to be more complicated with many more features and interactions between its features.

Given large-scale software product families (modeled using feature models), the main important question is how and what features should be selected for the next product that is going to be derived from this product family. The process of selecting the appropriate features for a product from the feature model is referred to as the *Configuration Process*. This process requires the consideration of many factors such as technical limitations, implementation costs, and stakeholders' expectations. Moreover, stakeholders are not always familiar with the structure of feature models and the available feature. In addition, stakeholders are often more comfortable to express their needs in terms of their goals and objectives. Therefore, in order to be able to select the best set of features based on the stakeholders' intentions and expectations, the software practitioners should understand the relations between the available SPL features and stakeholders' goals and intentions. It is not easy for the stakeholders to view a feature model and decide which set of features are the ones that they are interested in or which ones are the most beneficial and useful for their purpose. Even more, it is not enough to know what features exist and can be selected, what their interactions are and how they are able to perform their tasks, but it is also important for the stakeholders to know why these features essentially exist, interact and perform in the way that they do. Knowing the 'why' behind the existence of these feature model elements can easily speak to the intentions and objectives (goals) of the stakeholders [10].

A goal is defined as something that the stakeholders hope to achieve as a result of the development of a software product. In other words, it is the high-level objective of the business, organization or system owned, managed, or operated by the stakeholders [1]. Stakeholder goals have been widely captured through goal models within the requirements engineering domain [13][14][15]. Goal models are graph-like representations of the relationship between stakeholders' goals and their operational plans. They are often used to represent the realistic space of stakeholders' intentions and objectives [10]. Goal models are fundamentally built over three important concepts, namely *goals*, *soft-goals*, and *plans* (aka scenarios or tasks). Goals are objectives related to the functional aspects of the system. In contrast, soft-goals refer to quality attributes of the system. Furthermore, plans are ways to operationalize stakeholders' goals (Section 2.2).

The idea of employing goal models within the software product family has recently gained focus [12][15][17]. The proposed approaches have mainly attempted to use goal models for representing the variability of software products and transforming goal models into corresponding feature models. Although this idea provides the means for representing the features of a product family in terms of the stakeholders' objective, it confines variability to those defined based on the stakeholders' point of view (referred to as *essential variability*) and ignores technical and infrastructure variability (e.g., variability related to hardware, and operating systems). With this issue in mind, the main research questions that we are interested in addressing in this paper are: how are the most suitable features for a target application selected based on stakeholders' needs; how do the stakeholders' needs and goals relate with the available features within a feature model and variabilities (essential and technical), in other words, what is the relationship between the feature space and the intention space; and finally, how can a relationship between the stakeholders goals and the SPL features be formulated to select the best set of features for a target application of interest.

In this paper, we investigate how the most suitable set of features can be selected for the product configuration process by examining the stakeholders' needs and requirements gathered through a standard goal-oriented requirements engineering process. The stakeholders' intentions and goals are important within the software product line feature selection and prioritization process as they ensure that: 1) a complete and comprehensive set of initial features from the set of available features is selected (that can be passed into automated feature model configuration processes), which is due to the fact that we can make sure that all stakeholders' intentions, objectives and concerns are covered and addressed by the selected features; 2) irrelevant superfluous features are not included in the selected features, since features that do not correspond with at least one of the stakeholders' goals can be considered irrelevant for the target application and be not considered; and 3) the rationale behind the feature selection process is clear for the stakeholders. This becomes very important as stakeholders may not be able to clearly understand the utility of the selected features, but are able to analyze and see the importance of these features in relation to their objectives.

We propose explicit mappings to link features of a SPL feature model to the stakeholders' goals and objectives defined in goal models. We support this process by appropriate tooling and implementation technology (Section 3). These mappings help software practitioners move from the stakeholders' goals and expectations towards feature model selection decisions in such a way that a more desirable product is developed. Additionally, through the mapping mechanisms, practitioners can conduct reasoning on the stakeholders' preferences and objectives, and hence the most relevant features to stakeholder's objectives are configured automatically. Finally, by creating goal models and mapping them onto feature models, application engineers can communicate with the stakeholders based on their own jargon and abstract them away from realization details. To explain the proposed approach, we describe a case study (Section 2.1) and develop the case study using our proposed approach in Section 4. After acknowledging the related works in Section 5, we conclude the paper with a discussion on the advantages and limitations of our work as well as outlining future work.

2. BACKGROUND

2.1 An Illustrative Example

Before describing our proposed approach in detail, we introduce the case-study that is used for evaluating our approach. This benchmark case study consists of designing a family of *e-shop systems* that mainly supports ordering of products and shipment of the products to the customers. In our e-shop family, we assume that in all the e-shop systems, a common scenario is as follows: customers select some products and add them to their cart. They then perform the purchase process and the system verifies the customers' order and if the order is valid, the product(s) is/are shipped to the customers. To support this general scenario, the system should first show the products with their available quantities to the customers. It should also provide facilities for the customers to create an account, manage their account, and do online shopping. Also, the system should verify customer orders from the perspectives of validity and correctness. After approving the order, the system processes the payment. Finally, the products are shipped to the customers along with the issued bill. Different variations for the general scenario based on the specific stakeholders' functional and non-functional requirements can be defined. Additionally, each e-shop variant may include some additional scenarios in order to offer more to their customers. The e-shop family case-study has been used in both SPL [3] and goal-oriented requirements engineering [12]. We adopted the description in these resources and developed a case-study which combines the proposed case-studies in the literature. Due to space limitation, we only use a subset of the e-shop system, i.e., the "processing customer order" part.

2.2 Goal-Oriented Requirement Engineering

Goal-Oriented Requirements Engineering employs stakeholders' objectives and intentions called goals as a reference for eliciting, elaborating, structuring, specifying, analyzing, negotiating, and modifying requirements [9]. Goals define the desired states of affairs (i.e. descriptions of something that needs to be true in the world). However, existing goal-oriented researchers rephrase the desired state into the *generic activity* that corresponds to the desired state [9]. For example, the *Having Customer Order Processed* goal will be rephrased as *Process Order.* In order to be able to explore the exact and clear goals of the stakeholders and also understand the possible ways they can be satisfied, a formalization is required that would allow for better understandability, and intention traceability. To this end, goal models have been introduced and widely used, which are a controlled approach to organizing and structuring stakeholders' intentions in a graph-like representation [10]. Different variations of goal models have been introduced in the area of requirement engineering. We benefit from the *i* models* for building our goal models [11]. This framework is one of the most common frameworks for representing goal models. Goal models are fundamentally built over three important concepts, namely *goals, soft-goals*, and *plans* (aka scenarios, or tasks) [11]. *Goals* are objectives related to the functional aspects of the system (e.g., *Supply Customer Order* in the e-shop example is classified as a goal as it refers to a functional requirement). In contrast, *soft-goals* refer to non-functional or quality attributes of the system. For instance, *Customer Satisfaction* is a non-functional requirement that does not directly address system functional properties. Furthermore, plans are ways to operationalize stakeholders' goals. As an example in order to operationalize the *Determine Trustworthiness of Customer* goal, a possible plan would be to either *Check if is Returning Customer* or *Check Credit Score*. Goal models are often refined such that high-level goals are expressed through finer grained goals. This is achieved by using decomposition links. Decomposition links form tree-like structures that are equivalent to *and/or* trees, i.e., each parent node is broken down into smaller child nodes whose disjunction or conjunction will satisfy the parent. Moreover, in order to empower goal models with more qualitative relationships, contribution links with possible labels '+', '–', '++', and '– –' have been proposed [6]. The contribution links show to what extent a goal, soft-goal or developed plans contribute to the satisfaction of a soft-goal. The + (resp. –) and ++ (resp. – –) shows partial positive (partial negative) and full positive (full negative) contribution of the source goal to the target goal [6].

Figure 1 shows a goal model representing some of the stakeholders' expectations from the family of e-shop systems. The figure is a part of the developed goal model for the e-shop system family. This part of our goal model is adopted from [12]. It can be seen that one of the stakeholders' main goals is to be able to *Supply Customer Order,* decomposed into *Get Order*, *Verify the Order* and *Process Order*. The goal model shows how these high level goals can be refined and the related plans can be developed. Figure 1 also shows that the *Bill, Build, then Ship* and *Build, then Ship and Build* goals have partial negative and positive contribution to the *Customer Satisfaction* soft-goal, respectively.

So far, two main diagrammatic reasoning approaches, namely, *forward* [6] and *backward propagation* [8], have been proposed. The first reasoning approach on goal models is referred to as *the forward label propagation algorithm* [6]. This algorithm starts from lower level goals and works its way to the top goals. The approach defines some propagation rules on the relation among the goals. This indicates how satisfaction (denial) of source goal(s) is transferred onto higher level goals in the goal model. The propagation algorithm receives as input the level of satisfaction/denial (FS, PS, FD, PD) of lower level goals and plans and uses propagation rules to estimate the satisfaction of higher level goals. Thus, the algorithm can estimate to what extent high level goals are satisfied given the satisfiability of the lower level goals and plans.

On the other hand, *the backward label propagation algorithm* [8]

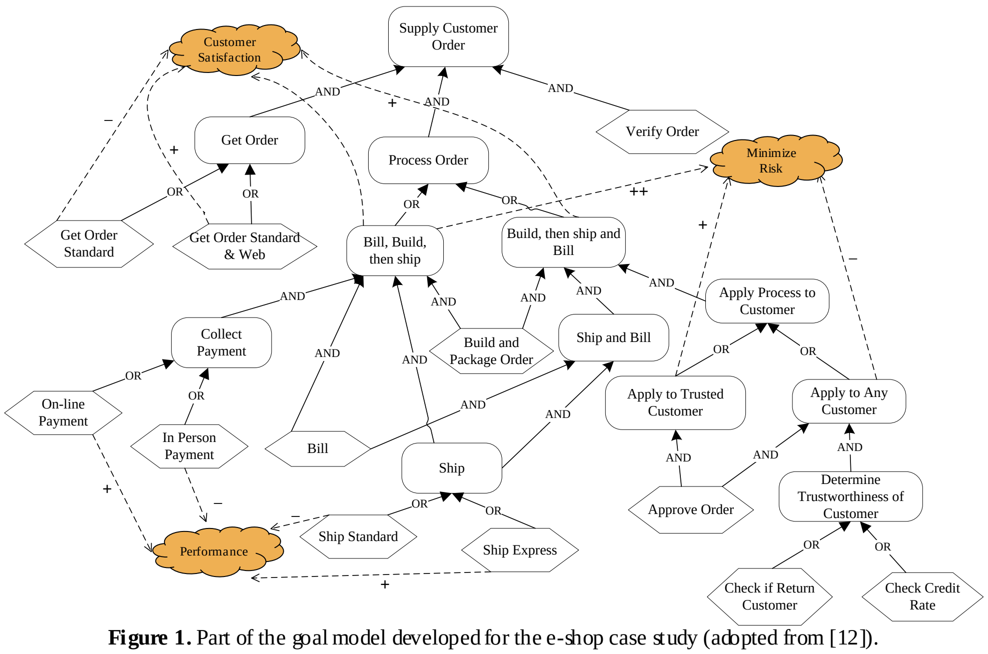


Figure 1. Part of the goal model developed for the e-shop case study (adopted from [12]).

accepts as input the desired degrees of satisfaction/denial (FS, PS, FD, PD) of a set of high level goals, and propagates these degrees throughout the goal model over the lower level goals and plans. The technique used to propagate the satisfaction level of higher levels to lower levels is a SAT technique. The details of the algorithm can be found in [8]. So, using the backward propagation algorithm, the stakeholders can select a set of high level goals as highly desirable, which will then be propagated through the goal diagram until the utility of all of the related lower level goals and plans have been calculated.

2.3 Feature Models

Features are important distinguishing aspects, qualities, or characteristics of a family of systems [7]. They are widely used for depicting the shared structure and behavior of a set of similar systems. To form a product family, all the features of a set of similar/related systems are composed into a feature model. A feature model represents possible configuration space of all the products of a system product family in terms of its features. Feature models can be represented both formally and graphically; however, the graphical notation depicted through a tree-like structure is more favored due to its visual appeal and easier understanding.

In a feature model, features are hierarchically organized and can be typically classified as: 1) *Mandatory*: a feature must be included in the description of its parent feature; 2) *Optional*: a feature may or may not be included in its parent description given the situation; 3) *Alternative feature group*: one and only one of features from the feature group can be included in the parent description; 4) *Or feature group*: one or more features from a feature group can be included in the description of the parent feature. In some case, the tree structure of feature models falls short at fully representing the complete set of mutual interdependencies of features; thus, additional constraints are often added to feature models and are referred to as *Integrity Constraints*. The two most widely used integrity constraints are: *Includes* – the presence of a given feature (set of features) requires the inclusion of another feature (set of features); and *Excludes* – the presence of a given (set of) feature(s) requires the elimination of another (set of) feature. Moreover, cardinality based feature models [3] (an extension of feature models) define *feature cardinality* and *feature group cardinality*. The former shows the number of instances of a feature in the final products, and the latter shows the minimum and maximum number of sub-features within the grouped feature that can be chosen for the final product. In the remainder of this paper, we use the cardinality based feature model definition.

Figure 2 depicts a part of the e-shop feature model, which we have developed for the e-shop case study. The feature model consists of two main features with the names *Front-store* used for providing facilities for customers to order a product and *Back Store* used for processing customer orders and taking care of payment along with shipment. As it can be seen, the *Check Credibility* feature, an optional feature, determines the trustworthiness of the customer. Moreover, as illustrated in the figure, for the *Payment Method* feature, at least two features of the existing features should be available in each product where one of them must be *Credit Card*. That is, for all possible configurations, in addition to the *Credit Card* feature, at least one of the *Debit Card, Money order, On-line Account to Account transformation, Cheque,* and *Cash* features must be chosen. Integrity constraints are defined – *New Customer* includes *Customer Verification* which means if the *New Customer* feature is chosen to be available in a product, the *Customer Verification* feature must also be selected for that product.

3. GOAL-FEATURE MODEL MAPPINGS

For mapping goal and feature models, we adopt an approach similar to the template-based approach proposed by Czarnezki et al. [3]. An overview of the mapping approach is shown in Figure 3. This figure shows the main products involved in the mapping process and also the activities that are performed to produce a preconfigured feature model. A family goal model represents the objectives of the family members with their relations and the feature model represents a hierarchy of features and the defined constraints between features. Through the developed mapping mechanism (discussed in Sect. 3.2), developers can annotate features with references to goals and create the mappings. Next, when a new product is needed, reasoning, based on the stakeholders' high

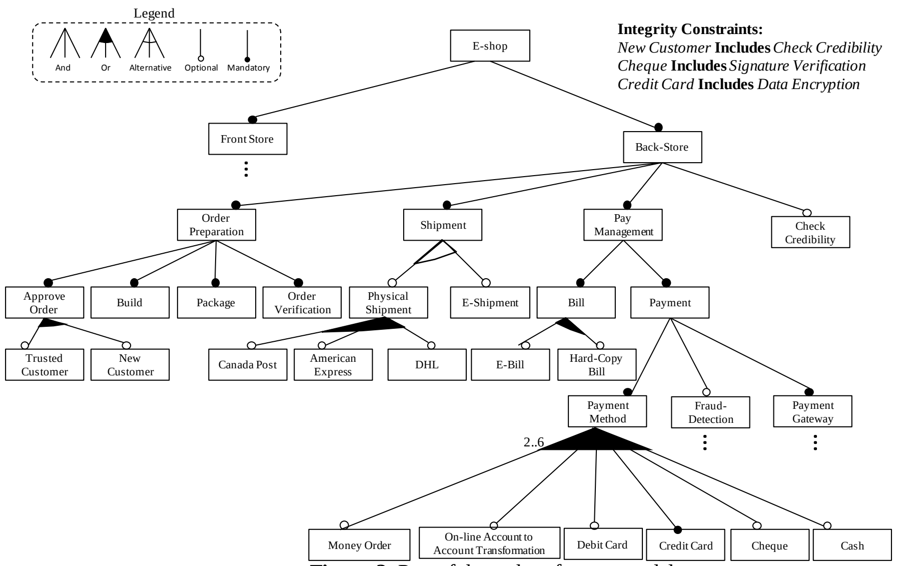


Figure 2. Part of the e-shop feature model

level goals is performed on the family goal model. Afterwards, an automatic pre-configuration of the feature model can be performed through the proposed algorithm introduced in Section 3.2 to produce the feature model which conforms to the stakeholders' goals. Our proposed approach ensures that the pre-configured feature model contains features which are based on the stakeholders' goals and preferences (soft-goals). Before describing the technical details of the mappings, we describe the steps that need to be performed in order to develop mappings by the domain engineers in the following.

3.1 Develop and Map Feature Model

In order to formulate the mappings, a requirements engineer needs to have both the feature model and the goal model available. Many techniques have been proposed by researchers in goal oriented community and SPL community to independently develop a goal model [7] [14] and feature model [21]. Additionally, feature model derivation from goal model has been explored by Yu et al [15] and Uno et al [4]. Developing a feature model from a goal model ensures traceability between features and stakeholders' intensions and facilitates the mapping process between the goal model and feature model. On the other hand, since we adopt Batory's definition of features [2] (i.e., features are incremental pieces of functionality), goal models (especially goals) can be used as a reference to define features and develop a feature

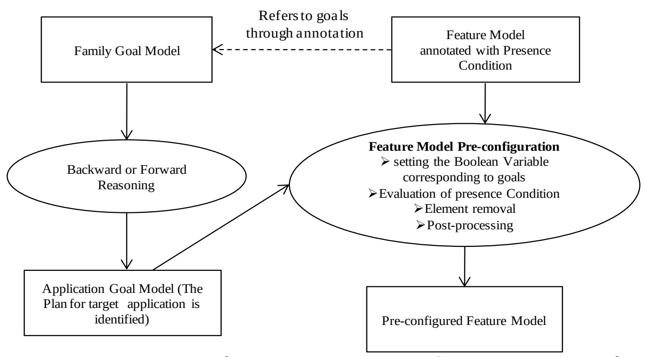

Family Goal Model
Refers to goals through annotation
Feature Model annotated with Presence Condition
Backward or Forward Reasoning
Feature Model Pre-configuration
➢ setting the Boolean Variable corresponding to goals
➢ Evaluation of presence Condition
➢ Element removal
➢ Post-processing
Application Goal Model (The Plan for target application is identified)
Pre-configured Feature Model

Figure 3. Overview of mapping process in application engineering for pre-configuring feature models (adapted from [3]).

model. Therefore, we assume the goal models have been developed by an existing approach. We propose steps to create a feature model based on the goal model and map them to the relevant goals.

- *Capture features related to tasks*, domain engineers need to examine low-level-goals (i.e., plans or task) and try to define features that can realize these tasks. For defining features for a task, the task, technical infrastructure related to the task and its environment constraints are considered to describe the features. For example, in the e-shop example, we have analyzed task *On-line Payment* as well as its required infrastructure and environment constraints. We identified some features including, *Debit Card, Credit Card, On-line Account to Account Transfer*. Furthermore, by analyzing required infrastructure and environment constraints (i.e. technical details) of these features, the features *Payment Gateway* and *Fraud Detection* are discovered. Feature *Payment Gateway* is OR-decomposed to features *Authorize.Net, Paraddata, Skipjack, VeriSign*, and *Payflow Pro*. Feature *Fraud Detection* is also OR-decomposed into *Authentication, Signature Verification,* and *Data-Encryption*. Finally, feature *Data Encryption* is analyzed and is XOR-decomposed into features *DES/3DES Algorithm, RCA,* and *IDEA*. Apparently, many features can be defined that reflect differences in the feature model on technical issues, about which some groups of stakeholders might necessarily not be concerned at that technical level. Note also that most of the features mentioned here are not shown in Figure 2 due to the space limit, although they are part of the feature model.
- *Define Parent features:* After extracting all atomic features related to the tasks in the goal model, domain engineers develop a feature model through iteratively grouping features with lower granularity to higher granularity by using existing relations in the feature model (i.e. OR, AND, Alternative) as well as the integrity constraints between features. In order to structure atomic features, domain engineers group features which are relevant to each other. Relevance of features is defined according to the relation between goals which they realize. Features realizing the goals which have the same direct or indirect parent can be con-

sidered to be relevant by domain engineers. Existing relations between goals (i.e., AND or OR-decomposition) are used to define proper relations between features in a feature model.

Define mappings between features and goals: This step can be performed either in parallel with the previous step or after the previous step is finished. Using the mapping structure (i.e. annotating features with goals that they realize) given in Sec. 3.2, domain engineers can create mappings between goal and feature models. At the end of this stage, the feature model and the mappings which map features to goals are created.

3.2 Mapping Infrastructure

Similar to the template-based approach [3], we employ the concept of *Presence Conditions* (PCs) to annotate features of feature models with the goals available in the related goal models. PCs are defined in terms of the goals and are evaluated based on their satisfaction degree (FD, PD, PS, FS). When a feature is annotated with a PC, the PC indicates whether features should remain in or be removed from the feature model during the pre-configuration process. PCs are expressed through the use of Boolean formulas over sets of variables, where each variable corresponds to a goal in the goal model. Since we consider features as functional increments, we only define PCs for the goals and task (not soft-goals). After reasoning on the goal model and determining the goal satisfaction/denial, the appropriate value is assigned to the variable. The variable is true if and only if the corresponding goal is labeled with FS, PS, and PD; and the variable is false if the corresponding goal is labeled with FD.

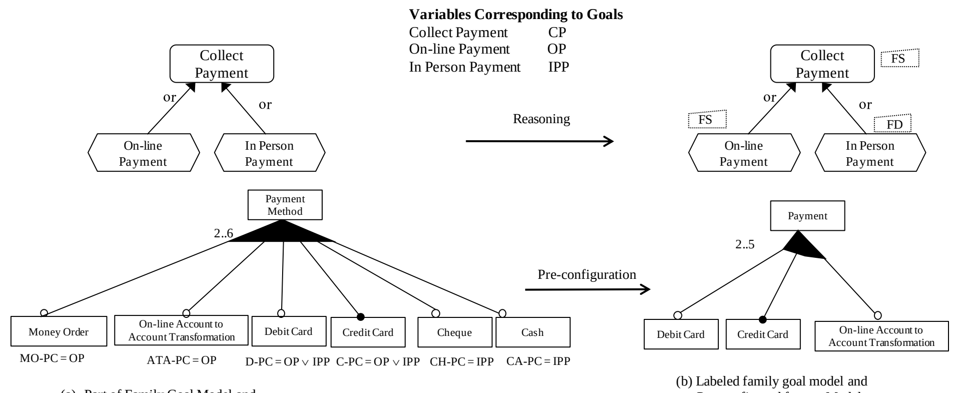
For example, consider Figure 4(a), which illustrates the *Collect Payment* goal along with the features realizing the tasks of the goal. A boolean variable is defined for each task (*CP* variable for *Collect Payment*, *OP* variable for *On-line Payment,* and *IPP* for *In Person Payment*). Next, according to goals which are mapped to features, PCs of the features are formulated as boolean expressions of variables corresponding to the goals. For example, the *On line Payment* and *In Person Payment* goals are mapped to the *Debit Card Payment* feature, so the feature's PC is expressed as $Debit\ Card\text{-}PC = OP \vee IPP$. The PCs are constructed automatically when domain engineers map features to goals through the use of the provided tooling support. Having features mapped to goals, the reasoning process over the goal model is able to label the goal model tasks based on the stakeholders' high-level objectives. As shown in Figure 4(b), the *On-line Payment* and *In Person Payment* tasks are labeled with Fully Satisfied (FS) and Fully Denied (FD), respectively. Due to the goal labels, the *OP* and *IPP* variables are valued to *true* and *false*, respectively. Consequently, the PCs of the features are evaluated and their final Boolean values are determined. After checking the rules, introduced in the next section, to ensure that feature model constraints have not been violated, the features *Money Order*, *Cheque* and *Cash* are removed from the feature model.

We should note that if a feature's PC is set to true then its corresponding sub-features' PCs are evaluated; otherwise if a feature's PC is false (i.e. it is determined that the feature should be removed from the feature model), then all its sub-features are removed except for sub-features which are involved in some integrity constraints such as *includes* for which suitable actions are defined. For example, let us assume the feature *Fraud Detection*'s PC is evaluated "false". So, the feature as well as its sub-features must be removed from feature model. However, the feature *Credit Card* includes the sub-feature *Data Encryption* of the feature *Fraud Detection*. On the other hand, the feature *Cheque* requires feature *Signature Verification* to validate the customer signature on the cheque. Hence, all other sub-features are removed except features *Data-encryption, Signature Verification* and the parent feature (i.e. *Fraud-Detection*). Moreover, existing feature relations in the feature model are removed when their corresponding features are deleted from the feature model. So, we do not need to annotate them or map them to the elements in the goal model. Since the mappings between features and goals are not one-to-one corresponding, PCs may become more complex logical expressions. When a feature is annotated with more than one goal, meaning the feature is used for realizing more than one goal, we define $\vee$ (Or) relation among corresponding variables of the goals. Also, one or more features may be used for realizing one goal. In such a case, all of them are annotated with that goal.

3.3 Feature Model Pre-Configuration

Through the use of the mapping mechanisms defined in the previous section, developers can map features to goals by annotating features with Boolean variables that correspond to the goals. Next, during configuration, the pre-configured feature model is generated automatically. The pre-configuration process is a *model-to-model transformation* where both the input and output models conform to the feature metamodel [5]. The process involves assigning *true* or *false* values to variables based on their corresponding goal labels (i.e., FS, PS, PD, FD), evaluating the PCs according to the variables and constraints in the feature model, and possibly employing some additional simplification processes.

After backward reasoning on the goal model, it can be seen that the goals with FD labels are not based on a high level objective of the current application stakeholders. Hence, features realizing



(a) Part of Family Goal Model and its corresponding Feature Model

(b) Labeled family goal model and Pre-configured feature Model

Figure 4. An example of mapping between goal and feature models where a reasoning algorithm. The algorithm selects the On-line payment plan based on the high-level objectives of the stakeholders. Consequently, the appropriate features are kept in the feature model.

such goals can be removed. That is, we considered the Boolean variable corresponding to each goal (hard-goal and plan) and each feature's PC is defined as a logical expression of these variables. Therefore, by assigning false values to variables whose corresponding goal is fully denied, we can evaluate the feature's PC. Then, features with values of PCs equal to false are candidates to be removed from the feature model. However, uncontrolled deletion of features may violate general constraints (e.g., removing a mandatory feature in an *And* grouped feature) or integrity constraints (e.g., removing one feature involved in an *include* relation while the other feature remains in the feature model). Therefore, during the pre-configuration process, we should make sure that these constraints are not violated. We discuss each of the constraints and describe the proper mechanisms (rules) for managing possible violations. All of these rules are checked automatically on the feature model and proper actions are adopted based on the case under investigation. In the remainder of this section, we call a feature a *Candidate Feature*, when the feature's PC is evaluated to *false*, which shows that the feature could be removed.

With respect to general constraints in the feature model, we discuss this for each relation (i.e., Or, And, Alternative) and type of features (i.e., mandatory or optional).

- *And, Or* and *Alternative* relations: If a candidate feature involved in an AND relation is a mandatory feature, the feature cannot be removed from the feature model, but it is labeled as a denied feature. For example in Figure 2, the *Order Preparation* feature cannot be removed even if its PC is evaluated to false, since it violates the feature model constraints. However, if a candidate feature is an optional feature, it can be removed from the feature model after checking its integrity constraints and group cardinality constraints. For instance, the *Customer Verification* feature in Figure 2 can be removed if its PC is evaluated to false. It still should be checked w.r.t the other constraints such as integrity and cardinality before the removal.
- *Group cardinality* <n_1, n_2>: If a feature is a *candidate feature* and the removal of the feature causes group cardinality violation (i.e. $n_1 > n_2$), first all siblings of the feature which are candidates to be removed are counted. If the deletion of the candidates violates the group cardinality constraint, they are kept in the feature model and are labeled as denied. For example, in Figure 4, removal of the *Cash* and *Cheque* features does not violate the feature cardinality constraint.
- *Feature cardinality*: feature cardinality does not impose any limitations on the removal of features from the feature model.

In all aforementioned cases, if a feature, which is removed from the feature model, is not an atomic feature, all of its children independent of their PC are removed from the feature model. In addition to the general constraints, integrity constraints may also affect the deletion of the candidate feature. These constraints include:

- F1 *includes* F2: if F1 is not a candidate feature and F2 is a candidate feature, according to the *includes* constraint, feature F2 cannot be removed from the feature model. Therefore, we keep feature F2 and label it as a denied feature. For example, in Figure 2, the *New Customer* feature includes *Check Credibility* feature. Assume the PC of the *New Customer* feature is evaluated to true and the PC of *Check Credibility* is evaluated to false. By applying the general rule and since it does not violate any of the general feature model constraints, it can be removed. Based on the integrity constraints, it cannot be removed and we should keep it and label the feature as denied.
- *F1 excludes F2*: independent of this integrity constraint, based on values of the PCs of these features and other constraints, a proper action is adopted. If the PC of each feature is false, the feature is removed after checking other constraints.

The *simplification step* is conducted on the feature model after the removal of the features in order to correct grouped cardinalities and remove the extra constraints such as *includes* and *excludes* when one of their features is removed. Additionally, for relations that only have one sub-feature and the other sub-features are removed; the parent feature is replaced with its sub-feature.

The pre-configuration algorithm can be summarized as follows:

1. *PCs evaluation*: The Boolean variables are evaluated to true or false based on their goals labels. Next, a depth-first algorithm is performed on the feature model and the PC of each visited feature is evaluated.
2. *Removal analysis*: For each features with their PCs evaluated to false, according to the situations that may occur, one of the above defined mechanisms is executed. At the end of this step, features with their PC evaluated to false are either removed from the feature model or are labeled as denied features.
3. *Simplification step:* Simplification of the feature model is done.

Feature model pre-configuration can be executed before the configuration processes such as staged configuration [5] and S-AHP [21]. In order to conduct pre-configuration the following steps should be done by application engineers: First, in the *capture stakeholders' high-level goals* step, application engineers communicate and understand stakeholders' needs by identifying their objectives. The family goal model is used as a reference model for communicating with the stakeholders and capturing their goals. In many cases, it would be impossible to fully satisfy all soft-goals [11]. For example, the full and simultaneous satisfaction of performance and security, goals is not possible. So, the desired level of satisfaction for each soft-goal is elicited by the stakeholders. Next, by setting the satisfaction level of high-level goals and executing the backward reasoning algorithm [6], the leaf goals (plans) are selected. Finally, the pre-configuration process is executed and the features, which are not based on the current stakeholders' objectives, are filtered out from the feature model. The features which are not based on stakeholders' goals, and have not been removed due to feature constraints are labeled as denied. Another scenario for the use of mappings is during the configuration process when an application engineer intends to further specialize the feature model. In this case, application engineers can trace back features to the goals in the goal model using the mapping infrastructure. Therefore, using the forward propagation algorithm, application engineers can find out the influence of removing a feature in satisfaction of high level objectives of the stakeholders.

3.4 Tooling Support

In order to provide tooling support, we have been customizing the *fmp2rms* plug-in [3], a tool that integrates the Feature Model plug-in with the Rational Software Modeler (RSM) and provides facilities to map feature models to implementation models. In our context, we reuse this tool for mapping feature models to goal models. That is, we developed a primitive prototype named *OpenME2fmp* that integrates the OpenME plug-in (an Eclipse plug-in used to develop goal models) and *fmp* (an Eclipse plug-in used to develop feature models). The *OpenME2fmp* utilizes the provided interface of *fmp2rms* and implements the pre-configuration algorithm. For the implementation of the Presence Conditions (PC), we consider boolean formulas in Disjunctive Normal Form. Thus, the only logical connector, that we support in PCs, is the Or relation ($\vee$). With respect to visualization of mappings, we present both feature models and goal models in the mapping space, by utilizing *fmp* and *OpenME*. We also provide the list view for the goal model which represents the variables corresponding to the goals and tasks. Domain engineers can switch between these views during the mapping process. Besides the mapping area, we provide an area where the mappings are presented (mapping View) from both the goal view and the feature view. That is, in the goal view, we list the goals and put the feature(s) which realize the goal as its sub-items and in the feature view. We also list the features and show the goals which are realized by the features as their sub-items.

4. DESCRIPTIVE CASE STUDY

In this section, we discuss our experience in applying the proposed approach to the e-shop case study. Since our objective is to evaluate and exemplify our approach – mappings between goal and feature models, we adapt and use an existing goal model in the literature [12]. Due to space limitation, we just show an application of our approach on a part of the case-study (Supply Order Management). As shown in Figure 1, the high-level objectives are *Support Customer Order, Get Order, Process Order,* and *Verify Order*. All goals identified on this level are common between different products of the e-shop family and required to be fully satisfied. However, different levels of satisfaction of soft-goals (i.e. *Minimize Risk*, *Customer Satisfaction*, and *Performance*.) for the products are identified. Relations between these high-level goals are represented with contribution links.

First, the feature model is developed through analyzing the goal model. For developing the feature model we follow steps introduced in Section 3.1. So, for each task, features required to implement the task are defined. By analyzing tasks w.r.t functionalities, environment constraints, and required infrastructure. The features shown in Table 1 are identified. Some of those features are common among different goals (e.g. *Credit card* is common between *On-line* and *In-person Payment*). Also, sometimes two or more goals may be realized just by one feature (e.g. *Check credit rate* and *Check if return customer* tasks are realized by the feature *Check credibility*). The *supply order process* goal model has 12 tasks, from which 47 features are derived.

Afterwards, the features are grouped with appropriate relation in feature model (i.e. And, Xor, Or). The grouping process start with features related to a task and then the relations in the goal model are used as direction for defining relation within feature model. For example, derived features from goal *On-line payment* are analyzed and different approaches of payment are grouped with the OR-relation to form a feature named *Payment Method*. According to the family description, a cardinality constraint on *Payment Method* is defined and mandatory and optional features are identified. Let us assume in our application domain all the stakeholders need at least two methods of payment and they want to have C*redit Card* payment as mandatory feature in all applications. So, we define cardinality constraints [2..n], which means that at least two should be selected for every product in the family. Additionally, features *Fraud Detection* is decomposed into *Data-encryption* and *Authentication*. *Payment Gateway* is OR-decomposed into the sub-features *Authorized.Net, para-data*, *Skipjack,* and *Versing Payflow Pro*. The features *Payment Method, Payment Gateway,* and *Fraud Detection* form an And-grouped feature named *Payment*. All these features are mapped to and thus annotated with the *On-line payment* task. By considering *in-person payment* task, the feature *payment method* is updated and three more sub-features (i.e. *Cash, Cheque,* and *Money Order*) added to this feature and mapped to the goal *In-Person payment*. Moreover, the feature *Signature Verification* is inserted to the sub-features of *Fraud Detection* and mapped to the *In-person payment* task. Finally, *Payment Gateway, Fraud Detection* and *Payment Methods* are mapped into *In-person Payment*. The feature model integrity constraints are defined as well. For example it, the feature *Credit Card* includes the feature *Encryption*. The process follows to develop feature model and map it to the goal model. The part of feature model is illustrated in Figure 2.

Table 1: The tasks in the supply customer goal model and related features.

Task	Related Features
Get Order Standard task	Paper Catalog, Cart (Item Order Management, Customer, Information Management), Fax, Phone
Get Order Standard &web	All the features of Get Order Standard task plus E-catalog (Search, Brows, Customview, Itemlist)
On line Payment	Credit Card, Debit Card, On-line Account to Account Transformation, Fraud Detection (Data encryption, authentication, DES/3DES Algorithm, RCA, and IDEA), Payment Gateway (Authorized.Net, Paradata, Skipjack, Versing Payflow Pro)
In Person Payment	Cash, Cheque, Money order, Fraud Detection (signature Verification), credit card, debit card.
Ship Standard	Ship preparation, shipping carrier (Canada Post)
Ship Express	All the features in Ship Standard task Plus Shipping Carrier (FedeX, UPS, USPS, Custom Shipping Gateway), Tracking System
Bill task	E-bill, Paper Bill
Approve Order	Approve Order

Check Credit rate	Check Credibility
Check if return Customer	Check Credibility
Build and Package	Order fulfillment (Physical item fulfillment, electrical item fulfillment, Service fulfillment)
Verify Order	Order management

Having developed the feature model, we can specify the high-level goals for each product and also the satisfaction level for each soft-goal. Then, the reasoning algorithm is applied to the goal model, based on which the pre-configured feature model is produced. As shown in Figure 4, for instance, after reasoning on the goal model, *On-line Payment* is labeled *fully satisfied* and *In-Person Payment* is labeled *fully denied*. Thus, *Credit Card, Debit Card,* and *On-line Account to Account Transformation* are kept in the feature model and *Cheque* and *Cash* are removed.

5. RELATED WORK

Goal-oriented software configuration has become an emerging area of research [13][14][15][16]. Here, we review some of the most relevant works on the theme of the paper. Liaskos et al [16] propose an approach which first identifies and understands configuration options of personal software; then they develop a goal model based on the configuration options and use it as a mediator between users and configuration options. Therefore, they automatically transform users' high-level goals into a configuration that satisfies the users' goals. Lapouchnian et al. [14] use goal models in designing and implementing autonomic systems which are able to select the best behavior according to the context. These systems are able to do self-optimization and self-healing. Our approach differs from these approaches in terms of the domain (i.e. our approach addresses the domain of SPL engineering which has a higher level of variability types and complexity). SPL covers both essential variability (i.e. Variability of the product family from stakeholders' point of view) and technical variability (Variability of the product family from the realization point of view). The goal model is used for representing and managing essential variability. Using goal models for technical variability diverges from their primary purpose – understandability for stakeholders. So, we used both goal and feature models for managing variability to facilitate the variability binding for stakeholders and developers.

Moreover, Yu et al employed goal models to develop *generic software* – "software solutions that can accommodate many/all possible functionalities that fulfill stakeholder goal" [15]. They extended the notations of the standard goal model with sequential, parallel, exclusive, non-system notations and define a set of heuristic rules which are employed to generate design models including feature models, state models, and component models. In [12], Yu et al. have proposed a dedicated tool for creating an initial feature model from a goal model by applying the similar transformation rules in [15]. The main issues of the approach are that first, they had to extend goal models with new notation elements and limit the structure of feature models to the structure of goal models. Although developers can change the feature model structure later, the main structure is still based on that of a goal model. Yet, we do not add any additional notation elements to goal models and our aim is to map feature models to goal models, not automate the generation of feature models from goal models.

Further, Antonia et al, [13] proposed ISARLPS, which defines a very high level process for developing feature models from goal-model represented by i* diagrams (i.e. Strategy Dependency [SD], and Strategy Rational [SR]). The steps in their approach are defined from a very high level and the approach is a transformation rather than mapping approach. Borba et al [17] also followed the approach similar to Antonio et aland define another set of heuristic rules for transforming goal models to feature models. Our approach is different from the above approaches in the policy which is applied to relations between feature models and goal models. We map feature models to goal models instead of transforming them into each other. First, we found out that goals are not the sole source of variability. Second, the structure of goal models is different from feature models. Third, goal models and feature models are designed to represent conceptually different information, and hence in many cases direct transformation does not provide rational results. Finally, we do not want to add new notation elements to the existing goal modeling language. As shown in Section 3.1, a feature model is created based on goal models and all rules defined by related research can be employed as suitable guidelines to formulate an initial feature model.

In SPLE, many approaches have been developed to map feature models to design and implementation models [3][18] [19]. Czarnecki et al [3] have proposed a template based approach which we adopted for mapping goal models to feature models. We had some other alternatives. For example, VML* proposed by Zschaler et al [18] is another approach for mapping features with solution artifacts. Another approach is used in FeatureMapper [19] that allows for mapping feature models to solution space models.

6. CONCLUSION AND DISCUSSION

We promote the consideration of early requirements information in the form of stakeholders' goals and objectives in the process of selecting the right features for a particular software product. Such an approach is simple yet effective for ensuring that the best set of features has been selected for a target application. The introduced approach can serve as a best practice guideline that provides a convenient way for capturing stakeholders' intentions, mapping them as concrete requirements into software features of the whole product line family, and feeding the process of the SPL configuration with a more accurate set of stakeholders' desired features. The main aim of this paper is to introduce the general idea of identifying the most suitable set of features of a software product feature model for the software practitioners. We argued that stakeholders' goals are among the suitable decision rationale for choosing the most suitable features to be included in the final product. We have proposed a framework to help select desired features to be included in the final software product.

As future work, we are going to enhance the support of non-functional properties in product lines by utilizing soft-goals. Currently, we are investigating to first define non-functional requirements based on soft-goals and quality properties and add them to the features through annotations. This is similar to the approach that we have already proposed in our feature selection technique called S-AHP [20].

7. REFERENCES

[1] Anton, A.I. Goal-based requirements analysis. *Proc. 2nd Requirements Engineering Conference* (1996) 136-144.

[2] Batory, D. Feature models, grammars, and propositional formulas. *Lecture notes in computer science 3714,* (2005) 7–20.

[3] Czarnecki, K. and Antkiewicz, M. Mapping features to models: A template approach based on superimposed variants. *Lecture notes in computer science 3676*, (2005) 422-437.

[4] Uno, K., Hayashi, S., and Saeki, M. Constructing Feature Models Using Goal-Oriented Analysis., *QSIC'09*, (2009), 412-417.

[5] Czarnecki, K., Helsen, S., and Eisenecker, U. Staged configuration using feature models. *LNCS 3154,* (2004), 266–283.

[6] Giorgini, P., et al., R. Reasoning with Goal Models. *Proc. Conceptual Model Conf. - ER'02* (2003), 167-181.

[7] Phol, K., Bockle, G., and Linden, F. *SPL Engineering: Foundations, Principles and Techniques*. Springer, 2005.

[8] Sebastiani, R., et al Simple and Minimum-Cost Satisfiability for Goal Models. (2004), 20-35.

[9] Van Lamsweerde, A. et al. Goal-oriented requirements engineering: A guided tour. *RE'01* (2001), 263.

[10] Yu, E. and Mylopoulos, J. From E-R to "A-R" — Modelling strategic actor relationships for business process reengineering. In *Proc. ER '94 Conf.* (1994), 548-565.

[11] Yu, E.S.K. Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. *RE'97* (1997), 226.

[12] Yu, Y., Leite, J.C.S.D.P., Lapouchnian, A., and Mylopoulos, J. Configuring features with stakeholder goals. *SAC'08* (2008), 645-649.

[13] António, S., Araújo, J., and Silva, C. Adapting the i* Framework for SPLs. *In Advances in Conceptual Modeling - Challenging Perspectives*. (2009) 286-295.

[14] Lapouchnian, A., Yu, Y., Liaskos, S., and Mylopoulos, J. Requirements-driven design of autonomic application software. CASCON'06, (2006).

[15] Yu, Y., et al From goals to high-variability software design. *Proc. 17th Int'l Conf. on Found. of Int. Sys* (2008), 1-16.

[16] Liaskos, S., et al. Configuring Common Personal Software: a Requirements-Driven Approach. *RE'05.* (2005), 9-18.

[17] Borba, C. and Silva, C. A Comparison of Goal-Oriented Approaches to Model SPLs Variability. *Proc. ER Workshops* (2009), 244-253.

[19] Zschaler, S., Sánchez, P., Santos, J., et al. VML* – a family of languages for variability management in SPLs. *SLE'10* (2010), 103-122

[20] Heidenreich, F., Kopcsek, J., and Wende, C. FeatureMapper: mapping features to models. ICSE Companion, (2008), 943-944.

[21] Bagheri, E., Asadi, M., Gasevic D., Soltani S., Stratified Analytic Hierarchy Process: Prioritization and Selection of Software Features, *14th Int'l SPL Conf.* (2010).

[22] Gacek, C. ed: Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. Springer , Heidelberg (2002).