Goal-Oriented Requirements and Feature Modeling for Software Product Line Engineering

Mohsen Asadi\textsuperscript{1,2}, Ebrahim Bagheri\textsuperscript{2}, Dragan Gasevic\textsuperscript{2}, Marek Hatala\textsuperscript{1}

\textsuperscript{1}School of Interactive Arts and Technology, Simon Fraser University, Canada
\textsuperscript{2}School of Computing and Information Systems, Athabasca University, Canada
masadi@sfu, ebagheri@athabascau.ca, dgasevic@acm.org, mhatala@sfu.ca

ABSTRACT
Feature Models encapsulate functionalities and quality properties of a product family. Leveraging feature models for managing variability and commonalities of large-scale product families raises an important question: on what basis should the features of a product line 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 to explicitly mapping and bridging features of a product line to stakeholders’ goals and objectives. Herewith, we propose a novel approach to automatically pre-configuring 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.

Keywords
SPL Engineering, Goal Oriented Requirements Engineering, Feature Model, Goal Model, Configuration

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 [11]. In SPLs, characteristics of a software system including functionalities and quality attributes are represented by Features [11]. To form a product family, in the Domain Engineering process, all the features of a set of similar/related software systems are composed into a Feature Model (Section 2.3). Later, the Application Engineering process takes the developed formal models, and also the needs, requirements and expectations of the stakeholders into account in order to develop a suitable final product. 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 them.

Given large-scale software product families (modeled in feature models), the main important question is how and what features should be selected for the next software 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 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 the 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 such a way. Knowing the ‘why’ behind the existence of these feature model elements can easily speak to the intentions and objectives (goals) of the stakeholders [14].

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]. 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 [14]. 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).

In the problem under study, the stakeholders’ intentions and goals are important 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.
As discussed in Section 6, the idea of employing goal models within the software product family has recently gained focus [17, 21, 23]. 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 the stakeholders' needs; how do the stakeholders' needs and goals relate with the available features within a feature model, 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 software 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 (Section 4). Simply stated, we propose explicit mappings to link features of a SPL to the stakeholders’ goals and objectives with its required tooling support (Section 3). These mappings help software practitioners in moving from the stakeholders’ goals and expectations towards feature model selection decisions in such a way that a more desirable product is developed. Moreover, to explain the proposed approach, we describe a case study (Section 2.1) and develop the case study using our proposed approach in Section 5. After acknowledging the related works in Section 6, we conclude the paper with a discussion about the advantages and limitations of our work as well as outlining the 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. In the remaining sections of the paper, we refer to the parts of this case study to discuss the background, mapping approach, and steps of the proposed process. The example consists of designing an e-shop system that mainly supports on-line ordering of products and shipment of the products to the customers. In our e-shop scenario, we assume that customers select the 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.

The e-shop case-study has been used by both SPL [7] and goal-oriented requirements engineering researchers [17][18]. We adopted the description in these resources and developed a case-study which combines the proposed case-studies in the literature.

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 [13]. Recently, various research efforts in the domain of goal oriented requirements engineering have been made to formally present stakeholders’ goals and perform reasoning on goal models. In the following, we explain the representation and reasoning approaches that are used in our context.

2.2.1 Goal Model Representation

Goals define the desired states of affairs (i.e. descriptions of something that needs to be true in the world). For example, in the e-shop case, one of the desired states (i.e. goal) is Having Customer Order Processed. However, existing goal-oriented researchers rephrase the desired state into the generic activity that corresponds to the desired state [10]. Hence, 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 [14]. Different variations of goal models have been introduced in the area of requirement engineering. We benefit from * models for building our goal models [15]. 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) [15]. 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 it is Returning Customer or Check Credit Score.

Figure 1. Part of the goal model developed for the e-shop case study, adopted from [17].

Goal models are often refined such that high-level goals are expressed through finer grained goals. This is achieved 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. In addition, since goals are operationalized through plans, they are interrelated through means-ends links. Plans can also be refined using decomposition links. Moreover, in
order to empower goal models with more qualitative relationships, contribution links with possible labels ++, +, −−, −+, −−, +−, and −+ have been proposed [8]. The contribution links show to what extent the 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 [8].

Figure 1 shows a goal model representing some of the stakeholders’ expectations from the e-shop system. The figure is a part of the developed goal model for the e-shop system. This part of our goal model is adopted from [17]. It can be seen that one of the stakeholders’ main goals is to be able to Supply Customer Order, which decomposes into 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. Moreover, Figure 1 shows that the Bill, Build, then Ship and Build, then Ship and Build goals have partial negative and positive contribution on the Customer Satisfaction soft-goal, respectively. Sometimes, one plan may have been used to operationalize more than one goal. For example, the Approve Order plan is used in both Apply to Trusted Customer and Apply to Any Customer.

2.2.2 Reasoning on Goal Models
So far, two main diagrammatic reasoning approaches, namely, forward [8] and backward propagation [12] have been proposed. Both approaches use the same notations for goal models. In these approaches, the goal model is considered as a set of goal nodes \( G \) and relations \((G_1,\ldots,G_n)\rightarrow G\) [12]. For each goal \( G_i \), four distinct predicates are defined including \( FS(G_i) \) and \( FD(G_i) \) meaning there is (at least) full evidence that goal \( G_i \) is satisfied and denied, respectively; and \( PS(G_i) \) or \( PD(G_i) \) meaning there is at least partial evidence that \( G_i \) is satisfied and denied, respectively [8]. Relations described on goal models are and, or, +s, −s +D, −D, ++s, −−s, ++−, +−−, −−+. We explain the meaning of these relations [8]:

\[
G_2 \rightarrow G_1 \quad (\text{resp. } G_2 \rightarrow G_1) \quad \text{means that if } G_2 \text{ is denied, then there is some (resp. a full) evidence that } G_1 \text{ is denied. However, if } G_2 \text{ is satisfied, then nothing can be said about the satisfaction of } G_1. \]

The first reasoning approach on goal models is referred to as the forward label propagation algorithm [12], which is introduced by Giorgini et al. This algorithm starts from lower level goals and works its way to the top goals. The approach defines some propagation rules on the relation between 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 the propagation rules to estimate the satisfaction of higher level goals. Therefore, the algorithm is able to 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 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. 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 [11]. 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 is a means for representing the possible configuration space of all the products of a system product family in terms of its features [9]. 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. More specifically, graphical feature models are in the form of a tree whose root node represents a domain concept (e.g., a domain application) and the other nodes and leaves illustrate the features. Here, a feature is a concept property related to a user-visible functional or non-functional requirement (e.g., domain application task) modeled in a way to capture commonalities or possibly differentiate among product family variants.

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 cases, 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); Excludes: the presence of a given (set of) feature(s) requires the elimination of another (set of) feature.

Moreover, cardinality based feature models [7] (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 for providing facilities for customers to order the product and Back Store for processing customer orders and taking care of payment along with shipment. As it can be seen, the Customer Verification feature, an optional feature, determines the trustworthiness of the customer through either the Check [if it is] Returning Customer feature or the Check Credit Rate feature. Moreover, as illustrated in the figure, for the payment feature, at least two features out of the existing four 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, Cheque, and Cash features must be chosen. With respect to integrity constraints, the New Customer feature includes the Customer Verification feature 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 AND FEATURE MODEL MAPPINGS

For mapping goal and feature models, we adopt an approach similar to the template-based approach proposed by Czarnecki et al. [4]. 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 pre-configured 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. 3.1), 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-level goals is performed on the family goal model. Afterwards, automatic pre-configuration of the feature model can be performed through the proposed algorithm in Section 3.2 to produce the feature model which conforms to stakeholders' goals. 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, then all its sub-features are removed except for

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, the features’ PCs 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-PC = OP ∨ IPP. The PCs are constructed automatically when domain engineers map features to the 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 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 Cheque and Cash are removed from the feature model.

3.1 Mapping Infrastructure

Similar to the template-based approach [4], 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 existing feature relations in the feature model are removed when they are involved in some integrity constraints such as includes for which suitable actions are defined. 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 or (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.

3.2 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 [6]. 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 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. 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 with 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 Customer Verification feature. Assume the PC of the New Customer feature is evaluated to true and the PC of Customer Verification 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 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.


3.3 Tooling Support

In order to provide tooling support, we intended to customize the fmp2rms plug-in [5], 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 tool named OpenME2fmp (see Fig. 5) 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 (DNF). The only operation which we identified is the Or relation (∨).

Figure 5. A screenshot of our OpenME2fmp toolset.

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.

In order to support developers in using our proposal and to make use of goal-oriented requirements engineering for developing product families and for configuring final products, we extended the traditional software engineering development process. SPL development processes typically have two lifecycles: Domain Engineering and Application Engineering. In domain engineering, common assets, family reference architecture and the variability model are developed. Then, in application engineering, the common assets are reused and variability model is configured to produce a product of the family. The set of activities and phases are defined in each of these lifecycle defined in the following. The changes are mostly concentrated on the early phases of these two lifecycles (i.e. family requirements engineering and application requirements engineering). Hence, we explain these phases in detail and briefly outline the other phases.

4. GOAL-ORIENTED PRODUCT LINE DEVELOPMENT PROCESS

4.1 Domain Engineering Lifecycle

4.1.1 Product Line Scoping

This phase identifies the possible range of products for the product line, analyzes marketing benefits, and understands the high-level business goals of the organization and also the constraints of the product line development effort. After identifying the products and their description, the product line roadmap, as far as it is foreseeable, is created. Besides the general activities of the product scoping phase, we define a new activity to help identify each product objectives on the business level. Therefore, the products descriptions are read by searching intentional keywords such as “objective”, “purposes”, “intents”, “in order to”, and others alike. This is performed in order to discover high level goals (goal and soft-goals) [13]. Next, the same goals appearing in different products (i.e. goals with the same semantics and maybe also the same name) are identified by the domain engineers. Finally, a merged list of high level goals for all products are created based on analyzing the extracted high level goals.

4.1.2 Goal-based Domain Requirements Engineering

This phase identifies and precisely and unambiguously documents common and variable requirements within the product line. The approach we propose to capture the requirements is an extension of existing goal oriented approaches for a single system [10]. The approach receives the list of high level goals and soft-goals and produces a complete integrated goal model, a forest containing all possible goals of the products belonging to that family. The following steps are performed to form the family goal model.

Soft-goal analysis: the purpose of this step is to analyze soft-goals and refine high-level soft-goals to fine-grained soft-goals. As mentioned before soft-goals are used to represent non-
functional requirements. The top-down refinement strategy can be used to refine the soft-goals to more detailed ones. Soft-goals may receive many positive or negative contribution links from other soft-goals. These relations are identified and modeled in this step.

Identify variability concerns for goal decomposition: after identifying the high level goals of the family, we need to decompose each goal to lower level goals and finally concrete plans. Deciding on how to decompose a goal requires that domain engineers understand variability concerns which are relevant to the goal. The purpose of this step is to identify the main relevant variability concerns and background variability for each goal. Liaskos et al. [10] have proposed a framework for constructing goal-based variability frames. In their approach, they define a variability frame for a goal; the variability frame contains a number of variability concerns and background variability. Variability concerns were defined as “type of questions whose alternative answers results in alternative refinements of the original goal” [10]. They define a set of general variability concerns including:

- **Factitive:** this concern is related to the object(s) or being(s) that is/are resulting from realizing the goal. For example in our goal model the Bill goal has \{e-bill, paper-bill\}.
- **Process:** the concern indicates the instrument and means that is used to perform the generic activity implied by the goal. Moreover, the manner that the activity is conducted is determined with this concern. For example the payment can be done in different ways like pay by debit, credit, or cash.
- **Temporal:** the concern is relevant to the duration and frequency of the generic activity that realizes the goal. For example, activity check for messages can have different defined frequency such as every hour or every ten minutes [10].
- **Conditional:** the alternative conditions under which a generic activity realizes the goal, which can either be fulfilled or triggered, are considered by this activity. For example, consider the ship product activity which has two alternatives including ship either when the order is arrived or payment is done.
- **Extent:** this concern examines in what possible alternative ways the generic activity can be performed. For example, if we have the report records activity, then different alternatives include report (10%, 50%, 60%) of the records [10].

The aforementioned variability concerns may not be equally intuitive for all goals for which they are used. In addition to the generic concerns, domain engineers can recognize problem-specific variability concerns and variability concerns related to the family when examining the individual goals. The former represents unintentional variability of the context of execution of the products, and the latter shows some variability concern which may be imposed on the goal, due to family issues. Domain engineers analyze each goal from a different perspective and identify all possible variability concerns as well as background visibilities.

Goal decomposition: After identifying the variability concerns of each goal, goal decomposition follows for each root goal (high level functional goal). Domain engineers organize the order by which concerns can be applied to address variability of the current goal. Generally, in AND-decompositions, variability concerns of the parent goal are inherited to at least one of its sub-goals. That is, when we perform an AND-decomposition on a goal, each sub-goal receives variability concerns which are relevant to that sub-goal. On the other hand, in OR-decompositions, one of the variability concerns is addressed and the goal is OR-decomposed based on the possible variants of the concern. The other variability concerns are inherited automatically to all sub-goals of the parent goal. By considering these two decomposition techniques, Liaskos et al. [10] proposed a concern-based decomposition process for goal models. The process starts by labeling variability concerns of the root goal to unresolved. Next, domain engineers can select based on variability concerns to perform either AND-decomposition or OR-decomposition. In AND-decomposition, since it only divides the variability concern set to a number of sub-sets, concerns still remain unresolved. In OR-decomposition, however, just one concern is addressed by partitioning its domain and assigning each partition to one sub-goal. Then, each sub-goal is analyzed with respect to the variability concern and if the concern has only one domain element, the concern is labeled as solved for that sub-goal; otherwise, the concern is labeled as addressed for the sub-goal. Domain engineers repeat concern-based AND- or OR-decompositions until all concerns are resolved.

Analyze decomposition effect on the soft-goals: Having defined OR- and AND-decompositions in the previous step, domain engineers should analyze the impact of each sub-goal on the soft-goals. Hence, domain engineers analyze the effect of each sub-goal of each goal on soft-goals, and model it with suitable contribution links. As was mentioned in Section 2.2, we use the i* framework and notation, which provides various ranges over contribution links that can be used for this purpose.

After defining the goal model in terms of goals and soft-goals and their relations, functional and non-functional requirements of the family can be defined. In goal-based requirements engineering, tasks in goal models are examined and if tasks should be performed by system-to-be, tasks are considered as system requirements. Otherwise, if tasks are performed by environment, tasks are treated as assumptions. Thus, we can define functional and non-functional requirements of the family through the family goal model. Having the requirements, an analysis is performed to understand the common and variable requirements of the family.

Variability modeling (feature modeling): After developing the goal model and defining the requirements, domain engineers identify the system features and design a feature model corresponding to the features of the target domain of application. Since the goals in the goal model represent the desired state through the generic activities and tasks, and since we also adopt Batory’s definition of features [2] (i.e. features as incremental functionality), goal models (especially goals) can be used as a reference to define features and develop a feature model. Similar to [2], we define the steps to develop a feature model by using a reference goal model:

Capture the atomic features: In order to capture atomic features, domain engineers need to examine low-level-goals (i.e., plans or task) and try to define features that can realize these tasks. For each task in the goal model, either one or more than one feature is defined. For example, in the e-shop example, we have analyzed On-line Payment and defined three features, including, the Debit Card, Credit Card, and On-line Account to Account Transfer features. Moreover, some of the previous identified features may be used to realize the current goal.

Form grouped features and define the constraints: 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). 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 considered 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. We should mention that the (functional) goals in a family goal model may form a forest (i.e., more than one tree structure), but features in the feature model form a tree-like structure. Additionally, family goal models represent both design and run-time variability, while feature models just capture design variability. For these reasons, it is not guaranteed that the hierarchical relations between features (or, and, alternative) are exactly the same as hierarchical relations in goals models. The next step is to identify integrity constraints along with feature and group cardinalities. By analyzing features referring to their goals, integrity constraints, groups and feature cardinalities are defined.

Define the 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 given in Section 3, domain engineers can create mappings between the goal and feature models. At the end of this stage, the feature model and the mappings which map features to goals are created.

4.1.3 Domain Design and Realization
This phase aims at developing features (i.e., domain assets) and reference architecture of the family. In order to develop features, design, implementation, and testing are performed. In Domain Design, architects try to reflect the requirements along with their variability within the reference architecture [14]. Afterwards, designers use object-oriented analysis and design techniques and develop a detailed design of the common and variable assets based on the reference architecture. The Domain Realization and Testing stage implements the reusable assets and validates them. Domain realization is performed by the incremental implementation of reusable assets and then testing them. For realizing the reusable assets, one of the make/buy/mine/commission procedures are conducted [15]. Some of the values of concerns in each feature are assigned to concerns during this phase.

4.2 Application Engineering Lifecycle
The application engineering lifecycle aims at developing a target application for the stakeholders by reusing existing assets in the family. Therefore, application engineers should capture stakeholders’ objectives (goals and soft-goals) and bind the feature model variability with respect to the objectives. Thanks to family goal models and the mappings between the goal and feature models produced in the domain engineering lifecycle, application engineers can make sure that the selected features are based on stakeholders’ objectives. Besides, the feature model is pre-configured and its complexity and size is decreased. The main phases of the lifecycle are goal-oriented application requirements engineering, application design, realization and testing.

4.2.1 Goal-oriented Application Engineering
This phase captures the stakeholders’ high-level objectives (goals and soft-goals), and performs reasoning on the goal model to extract plans (tasks) which satisfy the stakeholders’ objective. Next, based on the stakeholders’ objectives, automatic pre-configuration of the feature model is conducted. In the capture stakeholders’ high-level goals step, application engineers communicate and understand stakeholders’ needs and requirements 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 [16]. For example, a full satisfaction of performance, security, maintainability, and usability goals in the same-time is impossible. That is, the desired level of satisfaction for each soft-goal and the preferences between the soft-goals are elicited by the stakeholders. Afterwards, by setting the satisfaction level of high-level goals and executing the backward reasoning algorithm [8], the leaf goals (plans) are selected. Next, 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. Finally, application engineers employ existing configuration processes [3,27,28] to further configure the feature model.

4.2.2 Application Design and Realization
Having configured the feature model and bound the variability, its corresponding implementations as well as the application architecture are instantiated. This is followed by the further adaptation and specialization of the reference architecture. Moreover, in this phase, reusable assets are complemented by application specific components to produce the final product. After creating the final product, the application is verified with respect to stakeholders’ objectives and needs through performing different testing.

5. CASE STUDY
In this section, we discuss our experience in applying the proposed approach to the e-shop case study. First, we analyzed the e-shop product line family and identified high-level objectives which are Have Front Shop, Support Customer Order, Minimize Risk, User Interface Usability, security, Manage Shopping, Inform Customer, Do Shopping, and Process 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 the soft-goals for the products are identified. Next, we identified relations between these high-level goals and modeled the identified relations. For example, Have Front Shop includes the Manage Shopping, Inform Customer, and Do Shopping goals. So, we defined the AND-decomposition between them. Instead of starting from scratch for identifying the goals of the e-shop family, we reused existing goal models developed by [17][18].

In the next phase (i.e., goal-oriented requirements engineering), we developed the variability frame (set of variability concerns) for each of high level objectives and then based on the process proposed by [10] decomposed the high level goals. For example, for the Process Order goal (Figure 1), we identified Factitive (type of products which are created – Bill, Customer Product), Process (method of process customer order), and conditional (Type of customer – Trusted Customer or New Customer) variability concerns. The final decompositions of the Process Order goal is in Figure 1. After the decomposition of high-level goals, we defined the contribution links. For example, the Bill, Build and Ship goal has a positive contribution and the Build then Bill and Ship goal has a negative contribution on the Minimize Risk soft-goal, respectively. The final family goal model is depicted in Figure 6.

Next, the feature model is developed through analyzing the family goal model. First, for each task features required to implement the task are defined. For example, Credit Card, Debit Card, and On-line Account to Account Transformation features are developed for the on-line Payment task. Also Credit Card, Debit Card features are reused and Cheque and Cash features are developed for the In Person Payment task. Next, these features are grouped with OR-relations to form the Payment feature. According to the family description, a cardinality constraint on Payment is defined. This constraint means that at least two and at most four features should be selected for every
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, the Credit Card, Debit Card, and On-line Account to Account Transformation are kept in the feature model and Cheque and Cash are removed.

6. RELATED WORK

Goal-oriented software configuration has become an emerging area of research interest [19][20][21][22]. Here, we review some of the most relevant works on the theme of the paper. Liaskos et al [22] 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 mediator between the users and the configuration options. Therefore, they automatically transform the high level users’ goals and preferences into a configuration that satisfies the users’ goals. Lapouchian et al. [20] 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 at runtime. 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). Yu et al employed goal models to develop generic software – “software solutions that can accommodate many/all possible functionalities that fulfill stakeholder goal” [21]. 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 [17], Yu et al. have proposed a dedicated tool to create an initial feature model from goal model by applying the similar transformation rules in [21]. 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 automatic generation for feature models from goal models. Our approach also supports group cardinality.

Further, Antonia et al. [19] proposed ISARLPS, which defines very high level process for performing goal-based SPL using * diagrams (i.e. Strategy Dependency [SD], and Strategy Rational [SR]). The domain engineering process includes: i) develop SD model, ii) get features from the SD model, iii) develop SR model, and iv) develop a feature model. They also defined a set of rules to generate the feature model from * models. 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 [23] also followed the approach similar to Antonio et al’s approach and define another set of heuristics rules for transforming the goal model to feature model. Our approach is different from the above approaches in a 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 in feature models. Second, the structure of goal models (forest) is different from feature models. Third, goal models and feature models are found out that goals are not the sole source of variability in feature 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 in feature models. Second, the structure of goal models (forest) is different from feature models. Third, goal models and feature models are
The main aim of this paper is to introduce the general idea of feature models. We argued that, based on stakeholders' desired features, we can select 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 feature models for the software practitioners. We had some other alternatives. For example, VML* proposed by Zschaler et al [25] is another approach to mapping features with solution artifacts like architecture or activity models. Another approach is used in FeatureMapper [26] that allows for mapping feature models to solution space models.

7. 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 in which appropriate techniques are available to help select desired features to be included in the final software.

As future work, we are going to enhance the support of non-functional qualities in product lines by utilizing soft-goals. Currently, we are investigating to first define the 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 [27]. Next, according to the desired level of the soft-goals, we determine the quality value of each non-functional property. Finally, because we intend to employ SPL for developing family of service-oriented systems, we can select the appropriate services for implementing each feature with regards to non-functional properties. In this case, we can also use S-AHP to prioritize features based on user preferences.

8. REFERENCES


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


