A Enhanced Feature Model for Software Product Line and Core Feature Extraction

1st GuanZhong Yang

Department of Software Engineering
Hunan University
Changsha, China
gzyang@hnu.edu.cn

2nd HaoMing Chang

Department of Software Engineering
Hunan University
Changsha, China
haomingchang@hnu.edu.cn

3rd ZeYa Mou

Department of Software Engineering
Hunan University
Changsha, China
mouzey@hnu.edu.cn

Abstract—Feature model is an important model for capturing domain requirements and managing commonality and variability. However, the traditional feature modeling method is insufficient in expressing the variability requirements, which is easy to cause ambiguity and fails to describe the features in detail. In view of these shortcomings, this paper gives a unified definition of features, feature attributes and feature relationships, proposes configuration vector and variant constraint to enrich variability expression capabilities, and establishes an enhanced feature meta-model for software product lines. This feature model can help domain analysts effectively organize the variable domain requirements. In addition, the commonality of the domain expressed by the feature model is the basis for the development of highly reusable core assets. We propose that heuristic strategies automatically identify and extract core feature to obtain domain commonality. Finally, we take the chronic obstructive pulmonary disease home care product line as an example to demonstrate our proposed method.

Index Terms—feature modeling; software product line; core feature; commonality analyze

I. INTRODUCTION AND MOTIVATION

Software product line engineering is a paradigm for large-scale development of software applications [1]. The general idea is to plan and design reusable components for reuse before developing the system, and to manage the common and variable parts of the components [2]. As part of Feature-Oriented Domain Analysis(FODA) [3], feature model(FM) has proven to be an effective way to describe product line commonality and variability.

The Chronic Obstructive Pulmonary Diseases(COPD) Home Care System is a product line we are developing. The purpose of this application is to manage and monitor the stable COPD patients in home or community. It is designed to provide patients with environment monitoring, signs monitoring, doctor consultation, health education, acute exacerbation warnings and behavioral interventions. We have used FM to perform requirement analysis and domain analysis. Fig.1 uses the basic feature model(BFM) to demonstrate part of the features of the *signs monitoring* in the system.

Traditional feature modeling methods do not separate the perspectives of different stakeholders. The product line of

COPD home care involves services provided to users, domain knowledge related to COPD, various data monitoring devices and complex implementation technologies. If stakeholders share the same feature model view, they will not be able to effectively focus the perspective. Though extended-feature model [4]–[8] enrich the expression of variability from different perspectives, there is no agreement on the definition of modeling elements, especially two kinds of configuration information: mandatory and optional. This can lead to inadequate expression of variability and even semantic conflicts. Therefore, the traditional feature modeling methods are difficult to effectively satisfy the domain analysis requirements of our product line.

In addition, the COPD home care product line is a customizable application for individualized patients. The patient's use scenario and the patient's condition will bring more variability to the system. In the early stage of development, accurately extraction of core requirements in established domain model is required so that we can distinguish between common and variable requirements and build the basic platform based on core requirements in the subsequent development to speed up the time to market.

Fig. 1. Example of basic feature model.

In response to the above questions, this paper made the following contributions:

- Introduce an enhanced feature model meta-model. We classify features to form different levels of abstraction, propose configuration attribute vectors to resolve semantic conflicts and enrich the expression of model variability.

- Based on our meta-model, a core feature extraction heuristic strategy is proposed. The purpose is to automatically analyze the commonality of product families on a consistent feature model.

The rest of this paper is organized as follows. Sect.II describes the related work. Sect.III introduces the meta-model modeling elements. Sect.IV describes the core feature extraction method. We use a practical case to verify our ideas in Sect.V. And we conclude our work in Sect.VI.

II. RELATED WORK

[7] introduced Forfamel, a rigorous conceptual model foundation that combines multiple feature modeling methods. A cardinality-based feature model(CFM) formally defined in [4]. CFM adds the concept of feature cloning, using feature-cardinality to represent the number of cloned features. Group-cardinality in CFM is used to express configuration constraint of variants in variation points. [3] considered that different features can be refined into the same feature, and proposed the concept of pseudo-feature. [6] classified feature into capability feature, operational environment feature, domain technical feature, and implementation technical feature. The FM was divided into four levels corresponding to feature classification which mapped to implementation domain.

Some works have been devoted to extracting core features. The concept of core feature proposed in [8]. [9] transformed feature model into binary constraint logic, automatically analyzed all valid product through the CSP solver and obtained the core features. [10] proposed an algorithm traverses BDD by backtracking method to determine the core features and dead features. Although the algorithm was very efficient, converting the feature model to BDD was time consuming and reached an exponential level.

III. THE ELEMENTS AND RELATIONS OF META-MODEL

Fig. 2 shows our meta-model. We will introduce the terms and concepts necessary to understand the meta-model.

Fig. 2. Enhanced feature model meta-model.

A. Feature and Feature Attribute

Definition 1(Feature): A Feature f consist of a set of logically relevant software requirements that provide each stakeholder with the ability to satisfy specific business demand. A feature is defined as a 4-tuple: $f=(N,D,A,R)$, where: $N(\neq \phi)$ is the feature name to uniquely identify a feature. D is an explanation for the feature. A is a set of feature attributes, which describes the context information of the feature in a FM. R represents the relationship between the features.

We extend and represent the existing modeling elements with feature attributes. Feature attributes are divided into two categories:essential attribute and extended attribute. They are defined by a 2-tuple: $A=(T,V)$, where T is a set of attribute types for the feature, V is the value of the corresponding attribute type. Attribute of a feature can be expressed as: $f_A.type = value$. There are four types of feature attribute. $T:=\{Classification, Configuration, VariantConstraint, Extension\}$

Classification attribute. We classify feature into functional feature(FF), technical feature(TF) and system feature(SF).

- *Functional feature(FF)* is used to describe the services provided by the system, which can be clearly expressed and visible to users and is a specific behavior of the system.
- *Technical feature(TF)* is used to describe the technologies that implement functional features or non-functional indicators. Such features are not visible to the user, but developers are more concerned with these.
- *System feature(SF)* is used to describe the hardware resources, implementation environment of the system to facilitate communication between hardware and software engineers.

This categorization allows developers to refine different abstract level as they perform domain analysis and helps to separate the perspectives of different stakeholders. The expression $f_A.classification$ can be used to represent the classification attribute of feature f .

Configuration attribute. We attach optional and mandatory configuration information to the feature through feature attributes. Some domain feature modeling methods differ in their understanding of mandatory and optional feature. [5], [11], [12] gave an absolute definition of the mandatory feature, who considered that the mandatory features were the features appearing in all products in a product line, not related to its parent. However, [6], [7] interpreted it relatively, and considered that the mandatory feature must be selected when its parent feature was selected in the configuration process. We recognize and adopt the relative interpretation of the mandatory and optional, and introduces *core feature* to represent the absolute interpretation of mandatory feature. There are two drawbacks in the absolute definition of configuration attribute.

- It loses feature parent-child relationship contextual information.
- In the case that a feature has multi-parent features, the absolute definition is impossible to express the semantics

that child-feature's configuration attributes is different relative to different parent.

As shown in Fig.3(The specific features are represented by symbols.), feature $f1$ and $f2$ can be implemented by $f4$. The $f4$ is optional for $f1$ and mandatory for $f2$. When a feature have multi-parents, absolute interpretation of mandatory feature will cause ambiguous semantic. According to the definition of relativity in this paper, the configuration attribute of a feature may be different for different parent features, it is necessary to record these additional information which may change.

Fig. 3. Semantic conflict under absolute definition of configuration information.

Definition 2(Configuration vector): If the number of parent of a feature f is m and its parents are $(p1, p2, \dots, pm)$ from left to right in the feature diagram. Then C is a 1-dimensional vector of m components. $C := (bool_{p1}, bool_{p2}, \dots, bool_{pi}, \dots, bool_{pm})$

The component of vector C is boolean type, and $bool_{pi} = 1$ denotes that f is a mandatory feature relative to the i -th parent feature, otherwise it is an optional feature. We use $f_A.config = C$ to represent configuration attribute.

Variant constraint. This attribute represents the configuration constraint for all optional child-feature under a parent during configuration process. Similar to the group-cardinality in CFM. Fig.4 shows the feature *symptoms assessment*. In global initiative of COPD[13], we use $f2$ to evaluate the symptoms, and a comprehensive assessment is recommended using feature $f2$, $f3$ or $f4$ as supplement. Therefore, when selecting $f2$, at least one must be selected in $f3$, $f4$ and $f5$.

Definition 3(Variant constraint): $\langle n, m \rangle$ is an interval. The n is the lower bound and m is the upper bound. Let $k \geq 0$ be the number of optional child-features that satisfy $0 \leq n \leq m \leq k$. Variant constraint attributes of feature f indicate that if the feature is selected in configuration process, n to m optional features should be selected besides mandatory child-features. We use $f_A.variant = \langle n, m \rangle$ to represent configuration attribute.

Fig. 4. Example of variant constraint.

Extension attribute. This attribute is a feature extended attribute whose type is defined by the domain analyst. Such as adding the binding time and binding status of the feature.

B. Feature Relationships

Features form feature model with certain relationships. R is a set of feature relations and $R = R_{struct} \cup R_{constraint}$. $R_{struct} \subseteq f \times f$ represents the parent-child relationship of the feature. If feature $f1$ and $f2$ satisfy $(f1, f2) \subseteq R_{struct}$, then $f2$ is the parent of $f1$, and $f1$ is the child of $f2$. We extend three types of relations over structure relation, $R_{struct} = R_{decomposition} \cup R_{specialization} \cup R_{implement}$.

$R_{decomposition}$ describes the whole-part relationship between parent and child features. In Fig.1, $f1$ is decomposed into $f2$, $f3$, and $f4$.

$R_{specialization}$ represents that a feature can be specialized from different dimensions or aspects into more specific child-features. Child-features have the characteristics of parent, similar to inheritance in object-oriented, and they have special characteristics on the basis of inheritance. In fig.1, $f5$ is specialized into $f7$, $f8$, $f9$, $f10$.

$R_{implement}$ represents the technical implement or hardware support of the low-level features to the high-level features.

$R_{constraint} \subseteq f \times f$ represents the constraint between the same classification attribute features and $R_{constraint} = R_{req} \cup R_{excl}$. If $(f1, f2) \subseteq R_{req}$ indicates that $f1$ requires $f2$. If $f1$ is selected during configuration, then $f2$ also needs to be selected. This relationship is transitive. $(f1, f2) \subseteq R_{excl}$ indicates that $f1, f2$ cannot be bound at the same time, which is symmetry.

Fig. 5. Example of feature model structure.

Three classifications of features form three abstract levels of feature models. Fig.5 shows the basic structure of our FM diagram, where *root* is root of FM, *I-f* is the intermediate feature, *T-f* is the terminal feature, and *S-f* is the starting feature of the technical layer and the system layer. The above-mentioned structure relationships provide domain analysts with a way to construct a feature model. From root, the features are continuously refined through $R_{decomposition}$ and $R_{specialization}$. Then consider which technical features or system features are required to implement the service feature *T-f*. At this point, the technical feature or system feature is the *S-f* of the hierarchy, and then it can be considered whether these features can continue to be refined. By continuously iterating through this process, a FM diagram can be obtained. The relationship between $R_{decomposition}$ and $R_{specialization}$

makes the feature form the parent-child relationship of tree structure in the same layer, $R_{implement}$ connects T - f and S - f , thus connecting different abstract levels of FM, which becomes directed acyclic graph(DAG).

IV. ANALYSIS OF COMMONALITY

A. Core Feature Extraction Strategy

Definition 1(Core feature): Core feature is the feature that appear in all products which reflects the commonality of FM. If there are n valid products in the product line. Then core feature f_c are satisfied: $f_c \in \bigcap_{i=1}^n p_i$. Following rules are proposed to identify core feature:

Rule 1 The root feature in FM must be the core feature. Because the root appear in all products.

Rule 2 If there is a path starting from a core feature, and all the features on this path are mandatory for its parent, then these features are core feature. We call the core feature on this path directly accessible core features(DACF).

Rule 3 If there is a path starting from a core feature, and all the features on this path are linked by R_{req} , then these features are core feature. We call it the required transfer core feature(RTCF).

Rule 4 Indirectly accessible core features(IACF) that are easily overlooked.

Only extracting DACF and DTCF will result in incomplete core feature set. Fig.6 shows a special example (This figure is intended to make the example more intuitive, not the representation of our method.). Assume $f1, f_{11}, f_{12}, f_{13}, f_{14}$ as FF , $f2$ as TF . Feature f_{11}, f_{12} and f_{13} are all implemented by $f2$. Feature $f1$ is the core feature, f_{11}, f_{12} and f_{14} are optional, f_{13} is mandatory, and the $f2_{A.config} = (1, 1, 0)$. According to rule 2, only f_{13} is a core feature. However, because of $f1_{A.variant} = \langle 2, 2 \rangle$, one of the feature combination $\{f_{11}, f_{14}\}, \{f_{12}, f_{14}\}$ or $\{f_{11}, f_{12}\}$ must be selected. Therefore, f_{11} and f_{12} must be selected in the configuration process and $f2$ is a mandatory feature for f_{11} and f_{12} . Then, $f2$ which must be selected should be the core feature and it is IACF.

In more in-depth view, the main reason for the occurrence of IACF is that the variant constraints of core features make one of their optional child-features selected in the configuration process. It is precisely that these optional child-features have a common child and this child corresponded to these optional child-features is mandatory. Then this child that we called IACF must be selected. Therefore, a feature is IACF determined by its parents and its parents of the parents.

Fig. 6. Example of IACF.

We divide the structure in Fig.6 into three layers from top to bottom. When the following conditions are satisfied, the feature located in third layer must be the core feature.

Theory 1 The feature in first layer is a core feature.

Theory 2 The feature in the second layer is the optional child of the feature in the first layer, and these features all have the same child-feature, which is located in the third layer.

Theory 3 The feature located on the third layer is a mandatory feature relative to the features of the second layer.

Theory 4 Variant constraints of the feature on first layer make the feature on second layer must be selected.

Algorithm 1 get DACF set

Input: S_f

Output: S_{DACF}

```

1: function GETDACF( $S_f$ )
2:    $S_{DACF} \leftarrow \phi$ ;  $queue \leftarrow \phi$ ;  $wasIn \leftarrow \phi$ ;  $f_t \leftarrow \phi$ 
3:   for each  $f \in S_f$  do
4:      $queue.clear()$ ;  $wasIn.clear()$ 
5:      $queue.add(f)$ ;  $wasIn.add(f)$ 
6:     while  $queue \neq \phi$  do
7:        $f_t \leftarrow queue.poll()$ 
8:       if  $f_t \notin S_{cf}$  then
9:          $S_{DACF} \leftarrow f_t$ 
10:      end if
11:      for each  $f_c \in f_t.child$  do
12:        if  $f_c \notin wasIn$  &  $f_c.conf = (f_t, 1)$  &
            $f_c \notin S_{cf}$  then
13:           $wasIn \leftarrow f_c$ ;  $queue \leftarrow f_c$ ;
14:        end if
15:      end for
16:    end while
17:  end for
18:  return  $S_{DACF}$ 
19: end function

```

Next, we introduce the algorithms to extract DACF, DTCF and IRCF, then extract the core features completely.

B. Data Structures

We use the following data structures:

```

Feature{
  name:String
  child:list<Feature>
  parent:list<Feature>
  conf:map<Feature,Boolean>
  var:int[2]{n,m}
  req:list<Feature>
}

```

Where: *name* identifies a feature, *child* is the features child set, *parent* is the features parent set, *conf* is configuration attribute, *var* is variant constraint, *req* is required features.

C. Algorithm

Algorithm 1 gives a process of extracting DACF. The input is a core feature set S_f and output is S_{DACF} . S_{cf} is a

Algorithm 2 get RTCF set

Input: S_f
Output: S_{RTCF}

```
1: function GETRTCFSET( $S_f$ )
2:    $S_{RTCF} \leftarrow \phi$ 
3:   for each  $f \in S_f$  do
4:     GETRTCF( $f$ )
5:   end for
6:   return  $S_{RTCF}$ 
7: end function
8: function GETRTCF( $f$ )
9:   if  $f.req = \phi$  then
10:    return
11:   end if
12:   for each  $f_r \in f.req$  do
13:     if  $f_r \in S_{cf}$  then
14:       continue
15:     end if
16:      $S_{RTCF} \leftarrow f_r; \text{GETRTCF}(f_r)$ 
17:   end for
18: end function
```

global collection that holds all core features. The algorithm utilizes the breadth-first traversal of the graph, and sequentially searches for the mandatory child-features path starting from the feature f in the S_f . If f has multiple parents which are core features, as long as f is a mandatory feature relative to one of these parents, f is the core feature (line 12). Because of the mandatory path is formed.

Algorithm 2 gives the process of extracting the RTCF. Similarly, a core feature set is used as input, and the required transfer path starting from feature f in S_f is searched recursively, and all the features in the path are output. If a required feature of f is already the core feature, we ignore this feature to avoid repeated access (line 13-15).

Algorithm 3 gives the process of extracting IACF. As mentioned above, IACF must be a multi-parent features f_{mp} . So we take the multi-parent features set S_{mp} as input. The S_{mp} can be obtained by traversing the feature model once time (similar to algorithm 1), which is not discussed here. The feature f_{mp} is that IACF determined by the attributes and structure of its parent-feature set S_p and its parents of parents feature set S_{pp} . These features must satisfy the above four conditions. Firstly, the parent relationship is used to find the f_p which needs to satisfy *Theory2* and *Theory3* (lines 8-12). The f_p that satisfies the condition is stored in the S_p . In this set, we search for all f_p 's parent-feature f_{pp} which is core feature to satisfy *Theory1*, and store the valid f_{pp} in S_{pp} (lines 13-19). To satisfy *Theory4*, traverse the S_{pp} to get the number of all optional child-features N_{aoc} of f_{pp} , the number of child-features $N_{ocisp} \in S_p$, and the variant constraint value $[n, m]$ (lines 20-27). If $n \geq N_{aoc} - N_{ocisp} + 1$, it indicates that f_p must be selected in the S_p . Therefore f is the core feature.

Algorithm 4 introduces the main flow of extracting core feature set. The root feature is used as an input and output is

Algorithm 3 get IACF Set

Input: S_{mp}
Output: S_{IACF}

```
1: function GETIACF( $S_{mp}$ )
2:    $S_{IACF} \leftarrow \phi$ 
3:   for each  $f \in S_{mp}$  do
4:      $S_p \leftarrow \phi; S_{pp} \leftarrow \phi$ 
5:     if  $f \in S_{cf}$  then
6:       continue
7:     end if
8:     for each  $f_p \in f.parent$  do
9:       if  $f.conf = (f_p, 1) \& f_p \notin S_{cf}$  then
10:         $S_p \leftarrow f_p$ 
11:       end if
12:     end for
13:     for each  $f_p \in S_p$  do
14:       for each  $f_{pp} \in f_p.parent$  do
15:         if  $f_{pp} \in S_{cf}$  then
16:            $S_{pp} \leftarrow f_{pp}$ 
17:         end if
18:       end for
19:     end for
20:     for each  $f_{pp} \in S_{pp}$  do
21:        $n_{aoc} \leftarrow f_{pp}.optChildNum$ 
22:        $variant \leftarrow f_{pp}.var[0]; n_{ocisp} \leftarrow 0$ 
23:       for each  $f_c \in f_{pp}.child$  do
24:         if  $f_c \in S_p$  then
25:            $n_{ocisp}++$ 
26:         end if
27:       end for
28:       if  $variant \geq (n_{aoc} - n_{ocisp} + 1)$  then
29:          $S_{IACF} \leftarrow f$ 
30:         break
31:       end if
32:     end for
33:   end for
34:   return  $S_{IACF}$ 
35: end function
```

core feature set S_{cf} . When the core features are acquired each time using the above three algorithms, the newly discovered core features may cause other features associated with them to be transformed into core features. For example, a new core feature's mandatory child-feature or required feature is also the core feature. Therefore, we reuse the three algorithms to continuously synchronize the new core features to the S_{cf} until there are no new core feature. The set S_{temp} is used to temporarily store the core features acquired by a certain algorithm and use it as input to other algorithms. The set S_{mp} stores multi-parent features. In the inner loop, we alternately use algorithms 1 and 2 to obtain core features until there are no new ones (lines 4-17). In the outer loop, we judge whether there are new IACF in the multi-parent features. If new core features appear, it is necessary to go through the inner loop. If it does not appear, it means that the core feature set has been

Algorithm 4 get Core Feature Set

Input: f_{root} **Output:** S_{cf}

```
1: function GETCOREFEATURE( $f_{root}$ )
2:    $S_{temp} \leftarrow GETDACF(f_{root})$ 
3:    $S_{mp} \leftarrow FINDMULTIPARENT(f_{root}); S_{cf} \leftarrow \phi$ 
4:   while (true) do
5:     while (true) do
6:       if  $S_{temp} = \phi$  then
7:         break
8:       end if
9:        $S_{cf} \leftarrow S_{temp} \cup S_{cf}$ 
10:       $S_{temp} \leftarrow GETRTCFSET(S_{temp})$ 
11:      if  $S_{temp} = \phi$  then
12:        break
13:      else
14:         $S_{cf} \leftarrow S_{temp} \cup S_{cf}$ 
15:         $S_{temp} \leftarrow GETDACF(S_{temp})$ 
16:      end if
17:    end while
18:     $S_{temp} \leftarrow GETIACF(S_{mp})$ 
19:    if  $S_{temp} = \phi$  then
20:      return  $S_{cf}$ 
21:    else
22:       $S_{temp} \leftarrow S_{temp} \cup GETRTCFSET(S_{temp})$ 
23:       $S_{cf} \leftarrow S_{temp} \cup S_{cf}$ 
24:       $S_{temp} \leftarrow GETDACF(S_{temp})$ 
25:    end if
26:  end while
27: end function
```

obtained completely.

V. CASE STUDY

We applied the above FM to the COPD home care system. As shown in Fig.7, it is part of the FM diagram of our product line, which takes the core feature *sign monitoring* as the root feature and covers the examples above. We use feature attributes and feature relations to describe the whole model, and apply the algorithm to extract the core features of the examples. According to the algorithm, $\{f1, f2, f3, f4, f9, f11, f19\}$ is the DACF. $\{f14\}$ is the RTCF, and $\{f15\}$ is the IACF. The final domain model has 336 features, including 264 service features, 64 technical features, 8 hardware features and 47 constraints. We implement the core feature extraction method, and the FM is transformed into a file representation in XML format. After applying the algorithm, 101 core features are obtained, and the proportion of core features to total features is 30.05%. Among the core features, 91 are DACF, 8 are RTCF, and 2 are IACF.

VI. CONCLUDE

In this paper, an enhanced feature meta-model is proposed to solve the ambiguous semantics, enrich the expressive ability of variable requirements, and provide a hierarchical way to

Fig. 7. Feature model fragment of COPD home care system.

construct feature model. Then, a heuristic strategy is proposed to identify and extract the core feature set on the meta-model. We implemented and applied this method to the product line of COPD family care system. The next step is to implement a visualization tool for our modelling method.

REFERENCES

- [1] K. Pohl, G. Bockle, and F. J. V. D. Linden, "Software product line engineering: Foundations, principles, and techniques," *Proceedings of the First Intl Workshop on Formal Methods in Software Product Line Engineering*, vol. 49, no. 12, pp. 29–32, 2005.
- [2] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, Tech. Rep., 1990.
- [3] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "Form: A feature-oriented reuse method with domain-specific reference architectures," *Annals of Software Engineering*, vol. 5, no. 1, p. 143, 1998.
- [4] K. Czarnecki, S. Helsen, and U. Eisencker, "Formalizing cardinality-based feature models and their specialization," *Software process: Improvement and practice*, vol. 10, no. 1, pp. 7–29, 2005.
- [5] T. Tenório, D. Dermeval, and I. I. Bittencourt, "On the use of ontology for dynamic reconfiguring software product line products," in *Proceedings of the ninth international conference on software engineering advances*, 2014, pp. 545–550.
- [6] D. Fey, R. Fajta, and A. Boros, "Feature modeling: A meta-model to enhance usability and usefulness," in *International Conference on Software Product Lines*. Springer, 2002, pp. 198–216.
- [7] T. Asikainen, T. Mannisto, and T. Soinen, "A unified conceptual foundation for feature modelling," in *10th International Software Product Line Conference (SPLC'06)*. IEEE, 2006, pp. 31–40.
- [8] J. Peña, M. G. Hinchey, A. Ruiz-Cortés, and P. Trinidad, "Building the core architecture of a nasa multiagent system product line," in *International Workshop on Agent-Oriented Software Engineering*. Springer, 2006, pp. 208–224.
- [9] D. Benavides, P. Trinidad, and A. R. Cortés, "Using constraint programming to reason on feature models," in *SEKE*, 2005, pp. 677–682.
- [10] H. Perez-Morago, R. Heradio, D. Fernandez-Amoros, R. Bean, and C. Cerrada, "Efficient identification of core and dead features in variability models," *IEEE Access*, vol. 3, pp. 2333–2340, 2015.
- [11] J. Van Gurp, J. Bosch, and M. Svahnberg, "On the notion of variability in software product lines," in *Proceedings Working IEEE/IFIP Conference on Software Architecture*. IEEE, 2001, pp. 45–54.
- [12] A. Abbas, I. F. Siddiqui, S. U.-J. Lee, and A. K. Bashir, "Binary pattern for nested cardinality constraints for software product line of iot-based feature models," *IEEE Access*, vol. 5, pp. 3971–3980, 2017.
- [13] C. F. Vogelmeier, G. J. Criner, F. J. Martinez, A. Anzueto, P. J. Barnes, J. Bourbeau, et al., "Global strategy for the diagnosis, management, and prevention of chronic obstructive lung disease 2017 report. gold executive summary," *American journal of respiratory and critical care medicine*, vol. 195, no. 5, pp. 557–582, 2017.