IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 1, 2017, accepted February 27, 2017, date of publication March 9, 2017, date of current version April 24, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2680470

Binary Pattern for Nested Cardinality Constraints for Software Product Line of IoT-Based Feature Models

ASAD ABBAS[1], ISMA FARAH SIDDIQUI[1], SCOTT UK-JIN LEE[1], AND ALI KASHIF BASHIR[2], (Senior Member, IEEE)

[1]Department of Computer Science and Engineering, Hanyang University, Ansan 15588, South Korea
[2]Graduate School of Information Science and Technology, Osaka University, Osaka 565-0871, Japan

Corresponding author: S. U.-J. Lee (scottlee@hanyang.ac.kr)

This work was supported by the National Research Foundation of Korea through the Korean Government (MSIP) under Grant NRF-2016R1C1B2008624.

ABSTRACT Software product line (SPL) is extensively used for reusability of resources in family of products. Feature modeling is an important technique used to manage common and variable features of SPL in applications, such as Internet of Things (IoT). In order to adopt SPL for application development, organizations require information, such as cost, scope, complexity, number of features, total number of products, and combination of features for each product to start the application development. Application development of IoT is varied in different contexts, such as heat sensor indoor and outdoor environment. Variability management of IoT applications enables to find the cost, scope, and complexity. All possible combinations of features make it easy to find the cost of individual application. However, exact number of all possible products and features combination for each product is more valuable information for an organization to adopt product line. In this paper, we have proposed binary pattern for nested cardinality constraints (BPNCC), which is simple and effective approach to calculate the exact number of products with complex relationships between application's feature models. Furthermore, BPNCC approach identifies the feasible features combinations of each IoT application by tracing the constraint relationship from top-to-bottom. BPNCC is an open source and tool-independent approach that does not hide the internal information of selected and non-selected IoT features. The proposed method is validated by implementing it on small and large IoT application feature models with ''n'' number of constraints, and it is found that the total number of products and all features combinations in each product without any constraint violation.

INDEX TERMS Software product line, feature model, internet of things (IoT), cost estimation models.

I. INTRODUCTION

Software Product Line (SPL) approaches enhance the software development in context of low cost for deployment and time-to-market for product development. SPL contains number of products with same domain sharing common resources with variation of some features. To manage common and variable resources, SPL makes more effective and efficient methods for systematic reuse of existing resources [1]. SPL approaches consist of two major parts, domain engineering and application development. Domain engineering is the set of all possible resources, whereas, application engineering is the development of different products with variable resources according to the requirements [2]. Domain engineering process involves product line management which identifies the whole scope of SPL with all possible common and variable resources. Moreover, domain engineering is the platform of all common and variable resources according to requirements of SPL. Application engineering is based on artifacts of domain engineering according to end-user requirements. Product developers have to select variable resources from domain artifacts on the basis of requirements, however, common resources always are part of every product [3].

To manage the common and variable features of SPL, a tree structure of known as Feature Model, is commonly used in literature. The model consists of all common and variable features. The common features are mandatory features for all products of SPL. However, variable features are used according to the required specifications of product [4], [5].

Variable features are hard to manage in product line because of complex relationships and constraints among features. Predefined variabilities in feature model are, i) alternative features, where one and only one feature is selected from number of alternative features, ii) optional features, in which a feature may or may not be selected and iii) OR group, where at least one feature must be selected among features from the same group [6], [7].

Variability can be extended, customized or changed according to the specific purpose of the product [8]. In contrast to commonalities, management of variabilities is more challenging task for organizations in context of large and complex feature model. For development of IoT applications from feature model, organizations have to invest the effort, cost, time etc., to construct the features in advance without any initial market benefits. Adaptation of feature model for any organization is based on the total initial cost and benefits to estimate their own budget for specific family of IoT applications. By considering initial and final budget, organizations estimate the total cost and then take the decision whether they should adopt the specific product line or not,. Cost of IoT application product line can be estimated by finding the total number of IoT applications. Examining the total number of products is a complex task due to the nested constraints exist in IoT application product line.

Cost estimation can be made by Structured Intuitive Model for Product Line Economics (SIMPLE) and Constructive Product Line Investment Model (COPLIMO) by using total product numbers of SPL [9]. Manual analyses of total number of products become impractical in a large feature model, as number of products grow with increase in new features and relationships among them. To improve the accuracy of economic models and quality-based cost estimation of product line, exact number of possible products is an obligatory required parameter. After adaptation of product line, organizations require the combinations of features for application engineering. Finding combinations of features is another hard task due to the complex relationships exist among features for contextual variability management. Combinations of features enable it to determine the functional and non-functional attributes of each application. Addition and deletion of features from a feature model may effect the overall scope of the product line, which may increase or decrease the complexity of feature model.

Therefore, to overcome the difficulties discussed above, we have proposed a novel and effective framework that works in group wise combinations of feature model. Our approach finds the total number of applications as well as all possible combinations of various features without violations of any constraint. The approach is suitable for both small and large IoT-based feature model with complex relationships and back trace constraints between parents. To calculate the total number of applications of IoT based feature model, we propose Binary Pattern for Nested Cardinality Constraints (BPNCC) framework. In Order to show the correctness and effectiveness of our approach, we have applied the proposed approach on small and large IoT based feature models. BPNCC counts the total number applications of a feature model with nested constraints, by calculating and converting the number of possible combination in each group (alternative, optional, OR) into binary pattern (0s, 1s), where 1 indicates selected feature and 0 represents the non-selected feature. Furthermore, our algorithm BPNCC also determine the combinations of features for every individual product of SPL where all possible features represent an application.

The contribution of this work is to use the variable feature model in a systematic way to find the total number of IoT based applications with binary patterns of variable features. In BPNCC, variability feature model of binary combinations for selected and non-selected features are based on group cardinality, such as alternative, optional, and OR group. The beauty of this method lies in the fact that it is independent of any tool and also does not hide the internal information of binary combinations of IoT features.

The remaining paper is organized as follows. Section 2 describes the background of feature model, section 3 presents related works for SPL in IoT applications, section 4 discusses about the proposed framework BPNCC, section 5 shows experimental work by applying BPNCC over IoT feature model from SPLOT. Lastly section 6 presents conclusion and future directions of our work.

II. FEATURE-ORIENTED PRODUCT DEVELOPMENT

Quality of application with lower cost and time to market are the main issues in software development industry since the 1960s. SPL approaches address these problems by increasing the reusability of features in multiple products that fall in same scope [10]–[12]. Features are the visible functional properties of a software to stakeholders or end-users. Feature Oriented Domain Analysis (FODA) for software development is proposed by Kyo C. Kang in 1990 [13]. The objective of FODA is to reuse functional and architectural components of feature model. The Domain model shows the family of products and desired system can be developed after refining the domain according to particular requirements. Feature Oriented Reuse Method (FORM) is proposed by Kyo C. Kang 1998 for domain analysis of family of software that shares some common resources [14]. FORM gives analysis of common and variable resources for efficient reusability of features. Reusability of modules for software development enables the time-to-market and offers lower cost. Reuse-Driven Software Engineering Business (RSEB) is used in traditional software development process based on architectural reusability with some variation points. The SPL approach needs to enhance the reusability of features in multiple products. Therefore, the feature model architecture by using RSEB reusability techniques improve the reusability of features [15]. Cardinality-Based Feature Modeling (CBFM) has been proposed with notational extension of FODA and FORM, by defined cardinality of every feature that clearly present the dependencies, relationships, and variation points of SPL feature model [16]. Variation points address the

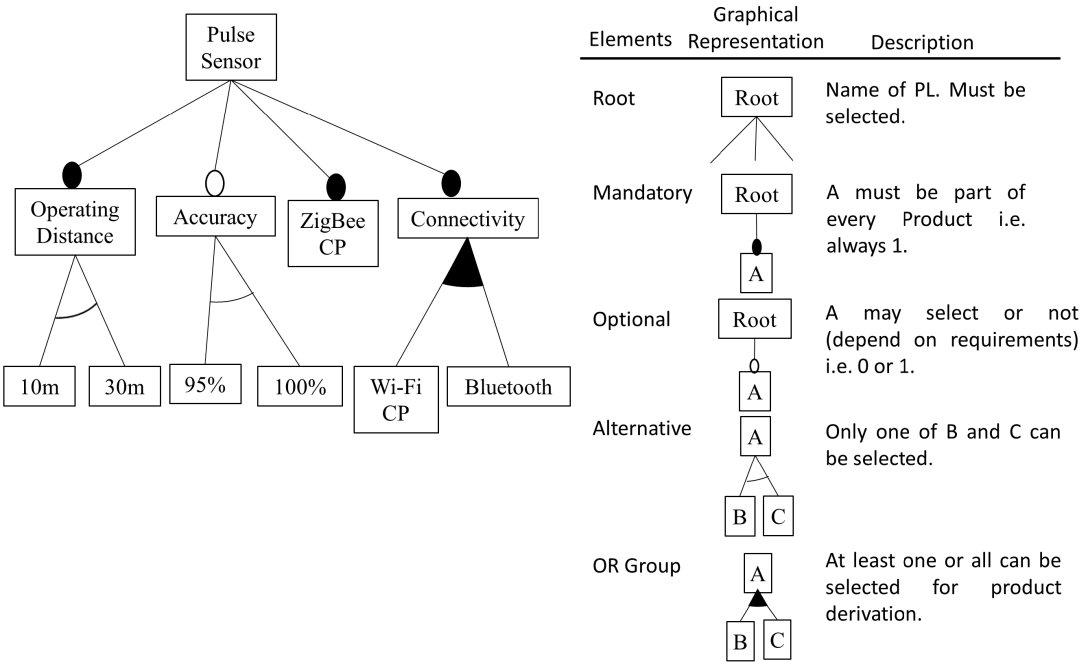

FIGURE 1. Pulse sensor motivational feature model.

variability of SPL families in application development. Dependencies between variation points effect the reusability of features because of bound functionalities. ConIPF Variability Modelling Framework (COVAMOF) tool is used for the traceability of variation points to remove the dependencies between features with defined cardinality notations of feature model [17]. Domain of SPL consists of all possible features which are used for development of products. The feature model is an effective approach to enhance the reusability of variable and common features. A feature model defines relationships and constraints between features for effective reusability [18], [19]. All features and their relationships are predefined in feature model.

The above Fig.1 shows feature model for a sensor module in an IoT application. The feature model shows Pulse Sensor as root node of model and having four child nodes. This Feature-Oriented approach for product development is implemented by four main processes [20]–[22].

1) Domain Analysis: This process captures all common and variable artifacts that are required for whole applications development, all resources are presented by feature model.
2) Domain Implementation: This process implements all common and variable features in functional form i.e. code.
3) Requirement Analysis: Capture and analyze all requirements from stakeholders and select suitable features from feature model for configuration of product.
4) Application development: Application is generated automatically from domain implementation and all configured selected features, according to end-user requirements.

Due to high presence of high risk of constraint violation, Effective and efficient techniques are required to variability management of feature model for development of different products of SPL as number of features and relationships increased in feature model [23].

III. RELATED WORKS

In SPL, variability management is a complex task due to complex relationships among features. Moreover, estimation of an exact total number as well all possible combination of features in a large scale feature model is complex and time consuming task. In recent research, there are scalability issues in SPL products due to the occurrence of constraint violations between relationships of features in final product development. The following sections summarize the related works in SPL and in IoT applications feature model briefly.

A. VARIABILITY MANAGEMENT IN SPL

Domain Knowledge Modeling Language (DKML) has been proposed to get the configuration information of feature model. Authors presented DKML about working for feature modeling and improved the configuration efficiency of SPL feature model. The conjunction between DKML and GenArch+ indicated the better performance for product derivations of SPL [24].

Cost estimation models such as SIMPLE and COPLIMO requires information from feature diagram to find the total estimated cost of SPL. These models need the total number

of variable and common features as well as total number of possible products. Authors have proposed the [25] algorithm based on NFT to calculate the total possible products. Their algorithm use same notations of VFD+ and NFT.

FeAture Model Analyzer (FAMA) framework has been proposed in literature for variability management in feature model [26]. FAMA supports to find out the valid FM product that satisfies all constraints to calculate the common aspects of feature which are used in number of products as well as calculates the total number of all possible products of feature model.

Another algorithm has been presented with one call execution to compute the total number of all possible products [27]. This algorithm is enhancement of NFT and VFD+ notations and is more efficient with respect to less computation and run time performance. This algorithm makes use of cardinality of leaf features to manage the constraints between features such as alternative, optional and OR.

Constraint Satisfaction Problem (CSP) has been used to manage the constraints of feature model [28]. It was described as an efficient way to manage the variability of feature model by using CSP models with Microsoft Solver Foundation (MSF) to find the product configurations and the total number of SPL products.

B. VARIABILITY MANAGEMENT OF IoT APPLICATIONS WITH FEATURE MODEL

Health-care applications have comprises of IoT are designed by Body Area Network (BAN) that have multiple contextual variability according to environmental selection. The main attribute of health-care applications is Quality of Service (QoS) that needs to be efficient due to patient runtime record. Proper and efficient modeling enables the increase in reusability of existing resources with high quality of health-care features. Feature modeling for variability management is used to manage the features and relationships between them in family of products. A BAN software is designed to entertain contextual variability in development of different applications such as health-care, transport, etc. By using feature modeling of BAN applications QoS can be achieved efficiently [29].

Embedded Software Ecosystem, where number of applications are interconnected to perform a specific task, but hard to manage due to the combinations of different features from multiple domains. Customarily, IoT applications are based on ESE where multiple features are interconnected with complex relationships. To overcome this problem, semantic reasoning and annotations are used for feature model variability management. In IoT application, features are modeled in contextual relationships to enable easy and efficient selection of features, according to end-user requirements [30].

IoT systems are usually combined with varied and interconnected devices that are handled by applications and varied with contextual environment and functional specifications. IoT devices can be used anywhere and connected with any other network device by using agent applications. Agent is middle part between IoT devices and cloud network. For the development of self-managed IoT systems, agents are more efficient due to distributed nature, self-adaptation, and context awareness. The variability management of agent based applications is complex due to the connectivity with IoT systems in different environment. SPL is used to manage the variability of agent applications for IoT systems. Common Variability Language (CVL) supports the variability notations of SPL feature model and is found to be effective for the management of common and variable features management [31].

The approaches discussed above for counting the total number of products of SPL work over a limited leaf nodes of feature model and also hide the inner information such as selected and non-selected features of feature model. Furthermore, under one parent but different feature cardinality, the groups need to be converted in new groups in order to differentiate them w.r.t to their cardinality, to calculate final total products. However, IoT applications require complete information of selected features to handle the contextual variability without hiding inner information. Moreover, the violation of constraints between relationships of various features are not fully handled in large feature models where nested constraints exist and payback in huge scalability issues. On the other hand, our proposed approach of BPNCC is simple and based on binary combinations of IoT application features that is applicable for unlimited back trace constraints and nested relationships between various features. In this work, we have considered only variability feature model for counting products, as commonalities are necessary to be the part of every application, and hence do not offer significant effect on the total number of products.

IV. BINARY PATTERN FOR NESTED CARDINALITY CONSTRAINTS (BPNCC)

The BPNCC is based on the binary combinations of features for constraints in leaf and parent nodes. The BPNCC is sequential approach to count the total number of products of feature model without violation of constraints. Moreover, this method counts the total number of products in large feature model, with n number of back trace nested constraints having zero constraint violation. Terminal features (leaf nodes) are required for product derivation as they are functional and visible to the end users. Terminal features are functional features that do not have further child features and are used to develop products of SPL. The actual functionality and product benefits are observed at terminal features. Non-terminal features represent the relationships among parents of all terminal features [6]. Therefore, we consider relationships between terminal features and non-terminal features of each group (alternative, optional, OR).

The framework the proposed novel and an effective approach to identify the total number of products of SPL is shown in Fig.2. SPL feature model consists of mandatory, optional, alternative, and OR groups. Mandatory features are always present in all products. However, variable features

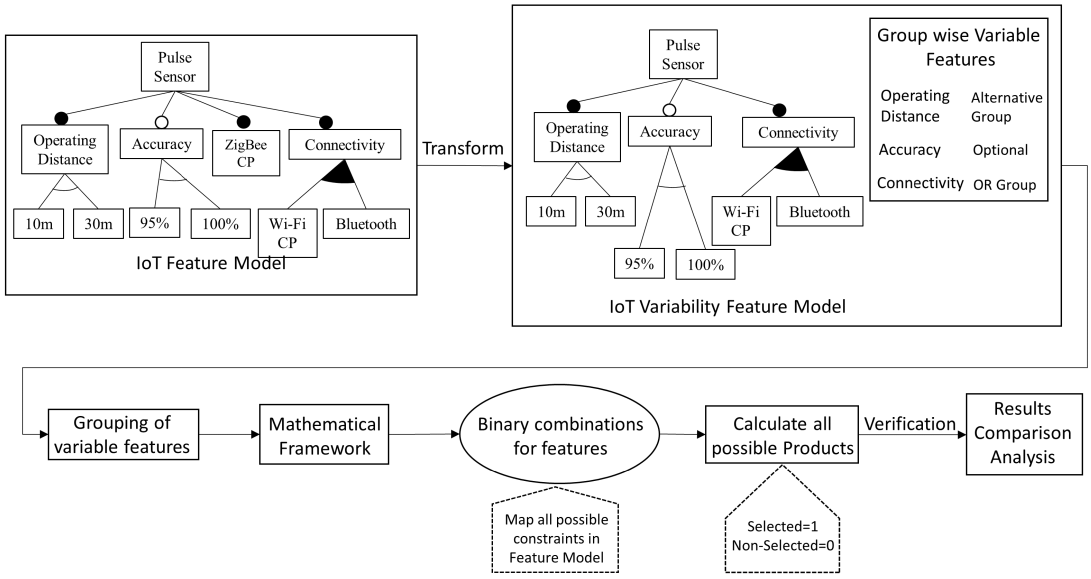

FIGURE 2. BPNCC proposed framework.

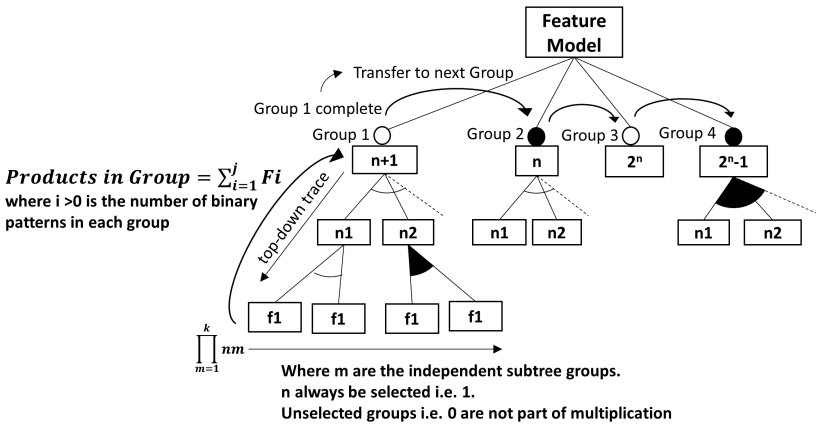

FIGURE 3. BPNCC mathematical framework.

differentiate the products in diverse selection of features. This BPNCC approach is based on five steps. The first Step is to transform the feature model while keeping only variable features and excluding all mandatory terminal features, as there is no any effect on total number of products. The second step is grouping of features that belongs to one constraint nested relationships. The third step uses mathematical framework to calculate the products with back trace tree structure.

with formulas respective to different variable groups. The fourth step makes binary combination of each group and subgroups of features. At last, the fifth step is to calculate the number of all possible products in feature model.

A. COMPUTE POSSIBLE SOLUTIONS

The primary task of our proposed approach is to model the variability features separately and identify all existing groups such as optional groups, alternative groups, and OR groups. To calculate the total number of products in feature model, we have proposed mathematical framework where all possible relationships of features of the following groups,

- Alternative Group: $n1 \times n2 \times \cdots \times nm$
- Optional Alternative: $n1 + 1$
- Mandatory OR: $2^n + 1$
- Optional group and OR: 2^n

Where n shows number of features, and m shows the last feature in a feature model.

Fig. 3 shows a 4 layered feature model with 9 terminal nodes and 6 non- terminal nodes. As can be observed in fig. 3, k shows the number of groups, n represents the number of terminal features in each group and doted lines shows the n leaf nodes. Group 1 consists of multiple nested relationships from top to bottom between nodes n1 and n2. Therefore, three possibilities exist, 1) neither will be selected because

```
Data: Input:
    1) R=Root Node
    2) N= Number of Groups
    3) NC=No. of Child of Root
    4) NCN=No. nodes of Current Node
    5) RCP= Parent to Child Relation.
       Mandatory=1, Optional=0, Leaf Node=0
    6) RCN=Relation between Child Nodes from Current Node
       Alternative, OR Group.
    7) LF= Leaf Nodes
    8) L=Length of Feature Model.
Initialization
while i<=N do
  if NC=0 then No Solution
  else if NC=1 then Define NCN and RCP
      if NCN=0 and RCP=0 then Two solutions
      if NCN=0 and RCP=1 then One Solution
      if NCN>0 then Define RCN
          if NCN=LF then compute all Solution to Root.
            [Combi]= Algorithm Module_2(L,RCN)
          if RCP>0 then Represent Mandatory, Discard all Zeros
      if NCN!=LF then Define L
          [Combi]= Algorithm Module_2(L,RCN)
          If RCP>0 then Represent Mandatory, Discard all Zeros
          Store all possible unique solutions.
      Generate Solutions RCN and RCP Relationship
    else if NC>1 then
        Deal all LF nodes R
          LF having relationship mandatory or optional with R
          RCP of LF with R Optional=Merge Solutions
          Depending upon the value of RCP, Refine all combinations
    Binary Combinations for each Group
end
Result:
Binary combination of each Group.
Total number of Products in Feature Model.
```

FIGURE 4. BPNCC algorithm: module1.

```
Data: Input:
    1) L= Group wise Length of Feature Model
    2) PR=Parent Relation with current node
    3) PRnew=empty
    4) C=2^L-1
    5) FV_idx=1
if PR=2 then
  Solve Subtree by using module 3
  [Combi, FV_idx]=Module_3(C, FV_idx,l)
else if PR=0 then
  Every Node have different relation with parent
  PRnew=Child-to-Parent Relation
  Solve subtree by using Module_3
  [Combi, FV_idx]=Module_3(C, FV_idx,PRnew)
Return
```

FIGURE 5. BPNCC algorithm: module2.

the parent is optional, 2) selection of n1, and 3) selection of n2 and same relationship in subtree. Second group has two possibilities as parent is mandatory, 1) selection of n1, and 2) selection of n2. Third is optional feature and it has two possibilities, selection or not selection. Fourth is mandatory OR group with three possible combinations, 1) selection of n1, 2) selection of n2, and 3) selection of both n1 and n2. Binary patterns are based on existing relationships between nodes. The multiplication operation is performed between cardinality of selected independent feature subtree under

```
Data: Input:
    1) combi=2^n-1 n are the number of groups
    2) CombiN=Combination of Current subtree
    3) PR=Parent child Relation
    4) nidx=Next index of current feature subset
    5) pp=Relation of current node with parent
    6) NC=Number of child of current node
    7) CN=Current Node
    8) LF=Leaf Node
if NC!=LF then R_Com=Define common relation
  if R_Com=0 then No common relation
  else if C_Leaf=0 then compute sum of column leaf nodes subtree
    if R_Com= Alternative then
      if PR=0 then Discard all combinations where sum<1
      if PR=1 then Discard all combinations where sum>1 or sum=0
    if R-Com=OR Group then
      if PR=0 then Retain all combinations
      if PR=1 then Discard all solutions having sum=0
    if R_Com=No combine relation then
      For each child node:
        R_C_C Define Relation of child node with current node
        NC_C_C number of child nodes of current node
        if NC_C_C=0 and R_C_C=1 and PR=1 then
          Retain combinations where sum>0
        if NC_C_C=1 then Recursive call
          [CombiN]=Modul_3(Combi, FV_idx, PR)
        Else Retain combinations where sum>0
      Update FV-idx: FV_idx+NChild
    else initialize start: FV:idx for every child node
        CPR=Relation of child node with parent
        if CPR=Alternate Relation
          if R_Com>0 then same relation with parent and PR2=CPR
            Initialize current_idx : 1
            L: define number of child nodes for given nodes.
            C: Generate all possible 2^L - 1 combinations.
            [Combi, FV_idx] = Algorithm_C(C,current_idx,1);
            Store Combi, L, FV_idx and FV_idx+L-1 in structure.
            Update FV_idx : FV_idx + L;
          else [Combi,FV_idx] = Algortihm_C(Combi,FV_idx,1);
        if CPR=OR Group then
          [Combi,FV_idx] = Algorithm_C(Combi,FV_idx,0)
        else [Combi,FV_idx] = Algorithm_C(Combi,FV_idx,CPR);
  Initialize Last : FV_idx;
  if PR2=Alternative Relationship
    if R_Com>0 the all nodes have same group relation with parent
      Merge all details into final CombiN
    Else determine location where sum>1
  Else if PR2=OR Group then Discard columns having sum=0
if Current Node=Leaf Node then
    if relation is mandatory then discard all where sum =0
CombiN: All remaining combinations till this point
Return
```

FIGURE 6. BPNCC algorithm: module3.

same parent. At parent of subtrees, the sum operation is performed between the values of all binary patterns. Finally, multiplication operation is performed on all group values to get the total number of products of feature model. Final equation for total number of possible products from fig. 3 defined as follows,

$$No.\ of\ Prod = \prod \left(\sum_{i=1}^{n} G_n \right) \implies \sum G_1 \times \sum G_2 \times \ldots \times \sum G_n \quad (1)$$

From Equation 1, G is group number, we can compute all possible solutions by finding the value from independent groups and using sum of values from every binary

pattern value of each group. Finally, all possible products are computed by multiplying values from all groups. We have developed BPNCC algorithm in order to get automatically the total number of configurations and all possible combinations of various features. Fig. 3 shows a model where we have designed every possible situation and notations of feature model. BPNCC Algorithm consists of three modules, Fig. 4 shows first module that takes all notations and cardinalities of parent to child node. Input of this module starts from root node to terminal node of each group with all notations and relationships between features and get the total number of products. This module works on top-to-bottom trace approach from root to terminal nodes and compute all possible combinations of given feature model by utilizing parent-child relationship. Two direct relationships exist between child nodes with root node are optional or mandatory. Further relationships such as alternative and OR group always exist between groups of feature model. In this module we have defined two direct relationships with root node and one top-to-down features and other relationships are based on cardinality of groups.

Figure. 5 shows the second module of BPNCC Algorithm that takes number of features in each group for correctness of cardinality constraints and convert into binary combinations with respect to cardinality value of each group.

Figure. 6 shows the third module of BPNCC Algorithm that finds all possible combinations of various features in binary form, such as 1 indicate selected and 0 indicate non-selected feature in final solutions of feature model. We have considered all possible cardinality constraints and notations in this module to make it general for every feature model.

Module 3 of BPNCC algorithm starts from root node to make the groups according to cardinality values. From first group input the numbers of terminal nodes for top-to- bottom trace and collect all cardinality information with each sub-group and independent nodes. It stores the cardinality values during top-to-down trace and generate all possible combinations in the form of 0s and 1s. If cardinality relationship is alternative between three features, three combinations will be generated by calculating the sum=1 for each combination. This operation is followed by all groups and finally merge combination from each group by using equation 1 to get all possible combinations of features.

V. EXPERIMENTAL WORK

BPNCC is runtime algorithm and we have implemented on MATLAB R2015b. Fig. 7 shows an example of nested constraint feature model that contains all possible variable groups in feature model and relationships among them. Manual calculation of all number of products is impractical, time consuming task and error prone. We have applied BPNCC approach on feature model and found the total number of possible products by using formulas for each group.

In Fig. 7, Group 1 (F1) is optional mandatory group and also have different constraints among child nodes of F1. For all groups Group 1, Group 2 and Group 3, we have calculated

TABLE 1. Group 1.

F1-1	F1-2	F1-3	Products in each Pattern
1	0	0	2
0	1	0	4
0	0	1	7
1	1	0	8
1	0	1	14
0	1	1	28
1	1	1	56

TABLE 2. Group 2.

F2	F3	Products in each Pattern
0	0	1
0	1	1
1	0	1
1	1	1

TABLE 3. Group 3.

F4-1	F4-2	F4-3	Products in each Pattern
1	0	0	2
0	1	0	4
0	0	1	7

the possible binary combinations as shown in Table 1, Table 2 and Table 3 respectively with

$$\begin{aligned}
\#\,\text{P in Group 1} &= \sum_{a=1}^{2^n-1} \text{G1}(a) \\
\text{G1} &= 2+4+7+8+14+28+56 \\
\text{G1} &= 119 \\
\#\text{P in Group 2} &= \sum_{a=1}^{2^n} \text{G2}(a) \\
\text{G2} &= 1+1+1+1 \\
\text{G2} &= 4 \\
\#\text{P in Group 3} &= \sum_{a=1}^{n} \text{G3}(a) \\
\text{G3} &= 2+4+7 \\
\text{G3} &= 13
\end{aligned}$$

where 'a' is the pattern number in each group. Selected features are indicated by 1 and non-selected features are indicated by 0. The total possible combinations in each group are calculated by using equation 1 as given bellow,

$$\mathit{TotalNumberofProducts} = 119 \times 4 \times 13 \implies 6188$$

To verify our approach, we have applied BPNCC on case study of real time different small and large IoT-based feature model which has been adopted from SPLOT to obtain the total number of products in each model. The selected adopted models have multiple nested constraints of relationships.

We have applied our proposed BPNCC approach on IoT middle-ware feature model, as shown in Fig. 8. We applied

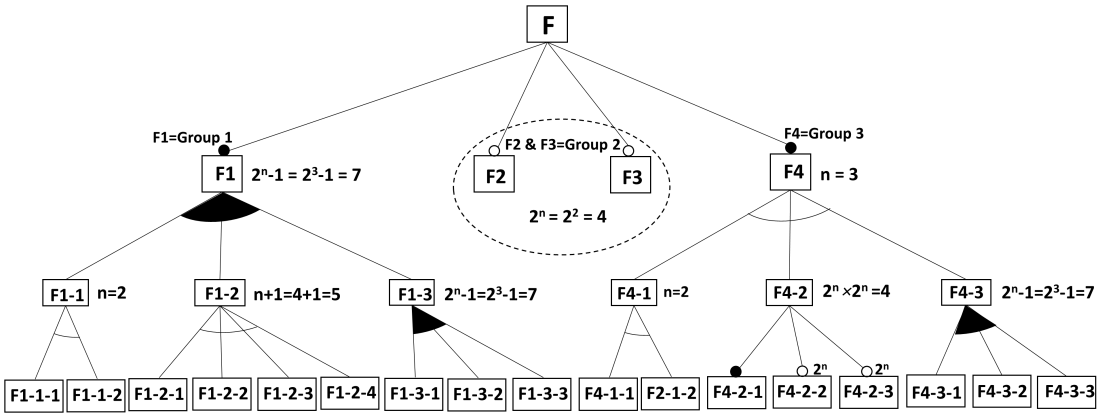

FIGURE 7. An example feature model with complex constraints.

TABLE 4. Binary combinations of sensor feature model.

Terminal Features	MotLike	Mobility	Movement	Distance	Temp	Reconfig	Data Delivery	Security	Context Awareness
#Solutions	3		7			4			
1	1	0	1	0	0	1	0	0	0
2	0	1	0	1	0	0	1	0	0
3	1	1	0	0	1	0	0	1	0
4	1	0	1	1	1	0	0	0	1
5	0	1	1	0	0	1	0	0	0
6	1	1	0	1	0	0	1	0	0
...	...	...	...	...	...	...	...	..	...
...	...	...	...	...	...	...	...	...	...
83	1	1	0	0	1	0	0	1	0
84	0	1	1	1	1	0	0	0	1

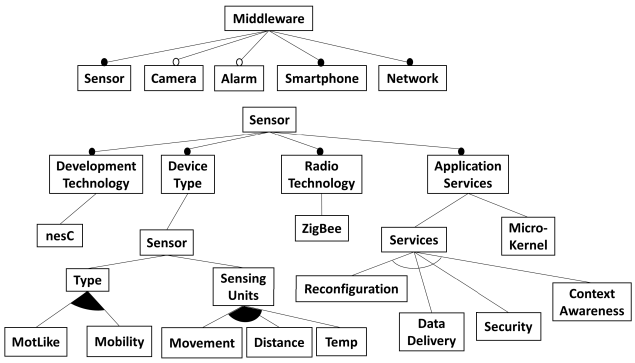

FIGURE 8. IoT feature model from SPLOT.

on single sensor subtree of Middleware and found the total number of products and solutions for all possible combinations of features.

A. RESULT DISCUSSION

From previous given case study of middle-ware sensor feature model we have computed total possible configurations and all features combinations of applications independently, as shown in Table 4. Total number of solutions are 84 and solution of features combinations are also same numbers as 84. Variation of numbers in both computation leads to the constraint violation. Therefore, same number shows occurrence of no constraint violation in both cases of total numbers and possible combinations of features. Our experimental results clearly indicate the total number of products can also be obtain by counting all solutions of binary combinations of features.

VI. CONCLUSION

SPL is an efficient approach for reusability of resources. Feature model is used to manage the commonalities and variable features of SPL. Cost estimation of complete SPL and individual applications is an important information for organization to adopt SPL. Cost estimation identifies the estimated budget that is required for development of all SPL products. Therefore, organizations need to calculate the budget before adaptation of to check whether the specific product line is under budget or not. Total number of products is primary parameter that is used in cost estimation models. Furthermore, combinations of all features for application development enable the selection and non-selection of features and also calculate the functional and non-functional attributes of each application. IoT applications have contextual variability in different environments that need proper modeling for reusability of variable and common features. Due to large scale variability in IoT applications, it is difficult to manage the variability and also hard to find the functional and non-functional attribute values due to implementations in different environments. In this paper we have

proposed BPNCC approach to calculate the total number of IoT applications from same domain and applied it on complex IoT feature models where the back trace nested relationship between features exist. BPNCC approach requires cardinality of each group of feature model and convert it into binary patterns to calculate all possible features selection in every group. Finally, it combines all groups to calculate the total number of products and solutions of all combination. Through detailed experimentation, we have proved 100% correctness of BPNCC with no constraint violation in complex constrained feature model. The validation of results is based on the fact that independently applied BPNCC for total numbers and total solutions of combinations of features are equivalent.

In future, we will work over the optimization of IoT based features selection by using binary patterns for each application according to the requirements of end users. We will optimize the functional and non-functional attributes of every product and optimize the minimum values such as cost, memory consumption and maximize the performance and quality according to the end user specifications.

REFERENCES

[1] K. Lee, K. Kang, and J. Lee, "Concepts and guidelines of feature modeling for product line software engineering," in *Proc. Int. Conf. Softw. Reuse*, vol. 231. 2002, pp. 62–77.

[2] H. Gomaa, "Evolving software requirements and architectures using software product line concepts," in *Proc. 2nd Int. Workshop Twin Peaks Requirements Archit. (TwinPeaks)*, May 2013, pp. 24–28.

[3] A. Metzger and K. Pohl, "Software product line engineering and variability management: Achievements and challenges," in *Proc. Future Softw. Eng.*, 2014, pp. 70–84.

[4] J. Lee and K. C. Kang, "Feature binding analysis for product line component development," in *Int. Workshop Softw. Product-Family Eng.*, vol. 3014. 2004, pp. 250–260.

[5] A. Abbas, Z. Wu, I. Siddiqui, and S. U.-J. Lee, "An approach for optimized feature selection in software product lines using union-find and genetic algorithms," *Indian J. Sci. Technol.*, vol. 9, no. 17, pp. 1–8, 2016.

[6] J. Guo, J. White, G. Wang, J. Li, and Y. Wang, "A genetic algorithm for optimized feature selection with resource constraints in software product lines," *J. Syst. Softw.*, vol. 84, no. 12, pp. 2208–2221, 2011.

[7] S. U.-J. Lee, "An effective methodology with automated product configuration for software product line development," *Math. Problems Eng.*, vol. 2015, Oct. 2015, Art. no. 435316.

[8] C. W. Krueger, "Introduction to the emerging practice of software product line development," *Methods Tools*, vol. 14, no. 3, pp. 3–15, 2006.

[9] H. P. In, J. Baik, S. Kim, Y. Yang, and B. Boehm, "A quality-based cost estimation model for the product line life cycle," *Commun. ACM*, vol. 49, no. 12, pp. 85–88, 2006.

[10] P. C. Clements and L. M. Northrop, "Software product lines: Practices and patterns," in *Sei Series in Software Engineering*. Reading, MA, USA: Addison-Wesley, 2001.

[11] I. Jacobson, M. Griss, and P. Jonsson, "Software reuse: Architecture, process and organization for business success," in *Proc. Israeli Conf. Comput. Syst. Softw. Eng.*, 1997, pp. 86–89.

[12] M. Sinnema and S. Deelstra, "Classifying variability modeling techniques," *Inf. Softw. Technol.*, vol. 49, no. 7, pp. 717–739, 2007.

[13] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Softw. Eng. Inst., Carnegie-Mellon Univ. Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-90-TR-21, 1990.

[14] 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," *Ann. Softw. Eng.*, vol. 5, pp. 143–168, Jan. 1998.

[15] M. L. Griss, J. Favaro, and M. d'Alessandro, "Integrating feature modeling with the RSEB," in *Proc. 5th Int. Conf. Softw. Reuse*, Jun. 1998, pp. 76–85.

[16] K. Czarnecki, S. Helsen, and U. Eisenecker, "Formalizing cardinality-based feature models and their specialization," *Softw. Process, Improvement Pract.*, vol. 10, no. 1, pp. 7–29, 2005.

[17] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch, "Managing variability in software product families," in *Proc. 2nd Groningen Workshop Softw. Variability Manage.*, 2004, pp. 1–11.

[18] A. Classen, P. Heymans, P.-Y. Schobbens, and A. Legay, "Symbolic model checking of software product lines," in *Proc. 33rd Int. Conf. Softw. Eng.*, 2011, pp. 321–330.

[19] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. L. Traon, "Assessing software product line testing via model-based mutation: An application to similarity testing," in *Proc. IEEE 6th Int. Conf. Softw. Test., Verification Validation Workshops (ICSTW)*, Mar. 2013, pp. 188–197.

[20] S. Apel and C. Kästner, "An overview of feature-oriented software development," *J. Object Technol.*, vol. 8, no. 5, pp. 49–84, 2009.

[21] A. Abbas, I. F. Siddiqui, and S. U.-J. Lee, "Multi-objective optimization of feature model in software product line: Perspectives and challenges," *Indian J. Sci. Technol.*, vol. 9, no. 45, pp. 327–334, 2016.

[22] A. Abbas, I. F. Siddiqui, and S. U.-J. Lee, "Goal-based modeling for requirement traceability of software product line," *J. Theor. Appl. Inf. Technol.*, vol. 94, no. 2, pp. 327–334, 2016.

[23] M. Mendonca, A. Wąsowski, and K. Czarnecki, "SAT-based analysis of feature models is easy," in *Proc. 13th Int. Softw. Product Line Conf.*, 2009, pp. 231–240.

[24] E. Cirilo, U. Kulesza, A. Garcia, D. Cowan, P. Alencar, and C. Lucena, "Configurable software product lines—Supporting heterogeneous configuration knowledge," in *Safe and Secure Software Reuse* (Lecture Notes in Computer Science), vol. 7925. Heidelberg, Germany: Springer, 2013, pp. 176–191.

[25] D. Fernandez-Amoros, R. H. Gil, and J. C. Somolinos, "Inferring information from feature diagrams to product line economic models," in *Proc. 13th Int. Softw. Product Line Conf.*, 2009, pp. 41–50.

[26] D. Benavides, S. Segura, P. Trinidad, and A. R. Cortés, "FAMA: Tooling a framework for the automated analysis of feature models," in *Proc. VaMoS*, 2007, pp. 1–6.

[27] D. Fernandez-Amoros, R. Heradio, J. A. Cerrada, and C. Cerrada, "A scalable approach to exact model and commonality counting for extended feature models," *IEEE Trans. Softw. Eng.*, vol. 40, no. 9, pp. 895–910, Sep. 2014.

[28] J. C. Navarro and J. Chavarriaga, "Using microsoft solver foundation to analyse feature models and configurations," in *Proc. 8th Euro Amer. Conf. Telematics Inf. Syst. (EATIS)*, Apr. 2016, pp. 1–8.

[29] A. Venčkauskas, V. Štuikys, N. Jusas, and R. Burbaitė, "Model-driven approach for body area network application development," *Sensors*, vol. 16, no. 5, p. 670, 2016.

[30] M. Tomlein and K. Grønbaek, "Semantic model of variability and capabilities of IoT applications for embedded software ecosystems," in *Proc. 13th Work. IEEE/IFIP Conf. Softw. Archit. (WICSA)*, Apr. 2016, pp. 247–252.

[31] I. Ayala, M. Amor, L. Fuentes, and J. Troya, "A software product line process to develop agents for the IoT," *Sensors*, vol. 15, no. 7, pp. 15640–15660, 2015.

ASAD ABBAS received the B.S. degree in information technology from the University of the Punjab, Pakistan, in 2011. Currently, he is pursuing his degree for M.S.-leading to the Ph.D. Program from the Department of Computer Science and Engineering, Hanyang University ERICA campus, Ansan, South Korea, funded by the Higher Education Commission of Pakistan, in 2014. His research interests include software product line, software requirement traceability, and IoT applications.

ISMA FARAH SIDDIQUI received the B.E. degree in software engineering and the M.E. degree in information technology from the Mehran University of Engineering and Technology, Pakistan, in 2006 and 2008, respectively. She is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Hanyang University ERICA, South Korea, funded by the Higher Education Commission, Pakistan. Since 2006, she has been with the Department of Software Engineering, Mehran University of Engineering and Technology, where she is currently an Assistant Professor. Her research interests include smart environment, semantic web, IoT, and big data.

SCOTT UK-JIN LEE received the B.E. degree in software engineering and the Ph.D. degree in computer science from the University of Auckland, New Zealand. He was a Post-Doctoral Research Fellow with Commissariat à l'énergieatomique et aux énergies alternatives, France. He has been with the Department of Computer Science and Engineering, Hanyang University, South Korea, since 2011. He is currently serving as the Head of the Department of Software for Emerging Technology. His research interests include software engineering, formal methods, web, and IoT. He is a member of the Korean Institute of Information Scientists and Engineers and the Korean Society of Computer and Information. He has served as an Editor, a Technical Chair and a Committee Member for various journals and conferences.

ALI KASHIF BASHIR (SM'16) received the Ph.D. degree in computer science and engineering from Korea University, South Korea. He held appointments with the Nara National College of Technology, Japan, the National Fusion Research Institute, South Korea, Southern Power Company Ltd., South Korea, and the Seoul Metropolitan Government, South Korea. In 2013, he joined the Graduate School of Information Science and Technology, Osaka University. His research interests include cloud computing, NFV/SDN, network virtualization, IoT, computer networks, RFID, sensor networks, wireless networks, and distributed computing. He is an active member of the IEICE and the IEEE Experts in Technology and Policy. He is serving as the Editor-in-chief of the IEEE Internet Technology Policy Newsletter and the IEEE Future Directions Newsletter. He is an Editorial Board Member of journals, such as the IEEE Access, the *Journal of Sensor Networks*, and the *Data Communications*. He has Chaired several conference sessions, gave several invited and keynote talks, and reviewed the technology leading articles for journals, such as the IEEE Transactions on Industrial Informatics, the *IEEE Communication Magazine*, the IEEE Communication Letters, and the IEICE Journals, and conferences, such as the IEEE Infocom, the IEEE ICC, the IEEE Globecom, and the IEEE Cloud of Things.